

AD-A196 113

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC FILE (1000) 1

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS
BEFORE COMPLETING FORM

1. REPORT NUMBER AFIT/CI/NR 88-123		2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) EXTENSIONS TO THE MULTILEVEL PROGRAMMING PROBLEM		5. TYPE OF REPORT & PERIOD COVERED PHD MS THESIS	
7. AUTHOR(s) JAMES THOMAS MOORE		6. PERFORMING ORG. REPORT NUMBER	
8. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: UNIVERSITY OF TEXAS- AUSTIN		9. CONTRACT OR GRANT NUMBER(s)	
11. CONTROLLING OFFICE NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) AFIT/NR Wright-Patterson AFB OH 45433-6583		12. REPORT DATE 1988	
		13. NUMBER OF PAGES 325	
		15. SECURITY CLASS. (of this report) UNCLASSIFIED	
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report) DISTRIBUTED UNLIMITED: APPROVED FOR PUBLIC RELEASE			
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) SAME AS REPORT			
18. SUPPLEMENTARY NOTES Approved for Public Release: IAW AFR 190-1 LYNN E. WOLAVER <i>Lynn Wolaver</i> 21 Feb 88 Dean for Research and Professional Development Air Force Institute of Technology Wright-Patterson AFB OH 45433-6583			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) ATTACHED			

DTIC
ELECTE
AUG 03 1988
S D

UNCLASSIFIED

PII Redacted

**EXTENSIONS TO THE MULTILEVEL
PROGRAMMING PROBLEM**

by

James Thomas Moore, B.A., M.B.A, M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 1988

ACKNOWLEDGEMENTS

I would like to thank Dr Jonathan Bard for his support and patience. The time he spent directing and guiding my efforts was invaluable. I am also grateful to Dr Melba Crawford, Dr J. Wesley Barnes, Dr Paul Jensen, and Dr Leon Lasdon. They taught me so much.

I appreciate the United States Air Force for giving me the opportunity to attend the University of Texas.

I thank Dr James E. Bennett, my classmate and friend, for his support and encouragement. I give special thanks to my parents for their encouragement and love. Finally, I want to thank Linda Stopkey. Her friendship and love were very important to me.

J.T.M.

The University of Texas at Austin

May 1988

111



Accession For	
NTIS CRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
TABLE OF CONTENTS	v
LIST OF TABLES	x
LIST OF FIGURES	xii
CHAPTER 1 INTRODUCTION	1
1.1 INTRODUCTION	1
1.2 OVERVIEW	2
CHAPTER 2 LITERATURE SEARCH	4
2.1 INTRODUCTION	4
2.2 RELEVANCE OF THE MODEL	6
2.3 PROBLEM FORMULATION	7
2.4 THE LINEAR BLPP	13
2.5 THE LINEAR MLPP	17
2.6 THE NONLINEAR MLPP	19

EXTENSIONS TO THE MULTILEVEL PROGRAMMING PROBLEM

Publication No.

James Thomas Moore, Ph.D.

The University of Texas at Austin, 1988

Supervising Professor: Jonathan Bard

The multilevel programming problem is extended by dealing with the integer and continuous forms of the linear bilevel programming problem. Algorithms are developed which solve the 0-1 bilevel programming problem, the continuous variable bilevel programming problem, and the mixed integer bilevel programming problem. The algorithms are tested, and an application is presented.

*Keywords: linear programming
computational, production planning, etc. (K)*

CHAPTER 3	NOTATION, DEFINITIONS, AND THEOREMS	21
3.1	INTRODUCTION	21
3.2	NOTATION	21
3.3	DEFINITIONS	22
3.4	THEOREMS	23
CHAPTER 4	AN ALGORITHM FOR SOLVING THE LINEAR 0-1 BILEVEL PROGRAMMING PROBLEM	26
4.1	THE BILEVEL PROGRAMMING PROBLEM WITH 0-1 VARIABLES	26
4.2	THE PARAMETRIC FORMULATION OF THE 0-1 BLPP	27
4.3	THE ALGORITHM	30
4.4	EXAMPLES	37
4.5	COMPUTATIONAL EXPERIENCE	43
CHAPTER 5	A BRANCH AND BOUND ALGORITHM FOR THE BILEVEL PROGRAMMING PROBLEM	48
5.1	DEVELOPMENT OF THE KUHN-TUCKER APPROACH	48
5.2	THE ALGORITHM	52
5.3	OTHER FORMS OF THE BLPP	58
5.4	EXAMPLES	62
5.5	COMPUTATIONAL EXPERIENCE	77
5.5.1	ALTERNATIVES EXPLORED DURING TESTING	79
5.5.2	RESULTS	81

CHAPTER 6	AN ALGORITHM FOR THE MIXED INTEGER BILEVEL PROGRAMMING PROBLEM	87
6.1	THE MIXED INTEGER BLPP	87
6.2	OBSTACLES TO ALGORITHMIC DEVELOPMENT	89
6.3	DEFINITIONS AND NOTATION	100
6.4	BOUNDS WHICH ALLOW THE SECOND TYPE OF FATHOMING	104
6.5	A MODIFICATION OF THE ALGORITHM WHICH SOLVES THE BLPP	109
6.6	AN ALGORITHM FOR THE MIXED INTEGER BLPP	115
6.6.1	THE BASIC ALGORITHM	116
6.6.2	DEVELOPMENT OF HEURISTICS	129
6.6.3	DEMONSTRATION	132
6.7	COMPUTATIONAL RESULTS	140
6.7.1	DETERMINING A BRANCHING VARIABLE SELECTION RULE	143
6.7.2	TESTING THE EIGHT ALGORITHMS	150
CHAPTER 7	THE BILEVEL PROGRAMMING PROBLEM AS APPLIED TO A PRODUCTION PLANNING PROBLEM	158
7.1	INTRODUCTION	158
7.2	FORMULATION AS A BLPP	161
7.3	EXAMPLE PROBLEMS	171

CHAPTER 8	CONCLUSIONS	181
8.1	INTRODUCTION	181
8.2	THE 0-1 BLPP	181
8.3	THE CONTINUOUS VARIABLE BLPP	183
8.4	THE MIXED INTEGER BLPP	184
8.5	APPLICATIONS	186
8.6	LESSONS LEARNED	187
APPENDIX A	FORTRAN COMPUTER PROGRAMS	189
A.1	INTRODUCTION	189
A.2	THE PROGRAM USED TO GENERATE RANDOM NUMBERS	189
A.3	COMPUTER PROGRAMS FOR THE 0-1 BLPP	192
A.3.1	PROGRAM FOR GENERATING RANDOM 0-1 BLPP	192
A.3.2	PROGRAM WHICH CREATES THE PARAMETRIC FORMULATION OF A 0-1 BLPP	198
A.3.3	THE ALGORITHM FOR THE 0-1 BLPP	204
A.4	XMP VARIABLE DEFINITIONS	221

A.5	PROGRAMS FOR THE BLPP	226
A.5.1	PROGRAM FOR CREATING RANDOMLY GENERATED BLPPs	226
A.5.2	PROGRAM WHICH ACCEPTS INPUTS OF A BLPP AND CREATES ITS EQUIVALENT KUHN-TUCKER REPRESENTATION	232
A.5.3	THE BLPP ALGORITHM	246
A.6	CODES FOR THE MIXED INTEGER BLPP	263
A.6.1	THE MIXED INTEGER BLPP RANDOM PROBLEM GENERATOR	263
A.6.2	THE ALGORITHMS FOR THE MIXED INTEGER BLPP	272
APPENDIX B	THE GENERAL MULTILEVEL PROGRAMMING PROBLEM	308
B.1	NOTATION AND DEFINITIONS	308
B.2	EXAMPLE B.1	314
BIBLIOGRAPHY		318

LIST OF TABLES

TABLE 4.1:	COMPUTATIONAL RESULTS	46
TABLE 5.1:	PROBLEM SIZES	82
TABLE 5.2:	COMPUTATIONAL RESULTS	83
TABLE 6.1:	THE EIGHT ALGORITHMS	132
TABLE 6.2:	BRANCH AND BOUND TREE DATA FOR EXAMPLE 6.2	139
TABLE 6.3:	TEST PROBLEM SIZE	145
TABLE 6.4:	CPU TIME (SECS) FOR RULES AT STEPS 3 AND 5	146
TABLE 6.5:	TEST PROBLEM SIZE	150
TABLE 6.6:	AVERAGE CPU TIME FOR ALGORITHMS	151
TABLE 6.7:	AVERAGE NUMBER OF VERTICES IN THE BRANCH AND BOUND TREE	152
TABLE 6.8:	AVERAGE NUMBER OF VERTICES TO FIND BEST FEASIBLE SOLUTION	152
TABLE 7.1:	PROBLEM DATA	172
TABLE 7.2:	PRODUCT DEMAND PER DOLLAR OF ADVERTISING	174
TABLE 7.3:	DEMAND, PRODUCTION, AND INVENTORY SCHEDULE OF PROBLEM (7.2)	177
TABLE 7.4:	DEMAND, PRODUCTION, INVENTORY, AND SUBCONTRACTING SCHEDULE OF PROBLEM (7.3)	179

TABLE 7.5:	RESULTS OF ALGORITHMS FOR PROBLEM (7.3)	180
TABLE B.1:	OBJECTIVE FUNCTION VALUES OF VARIOUS OPTIMA	317

LIST OF FIGURES

FIGURE 4.1:	FLOW CHART OF 0-1 BLPP ALGORITHM	34
FIGURE 4.2:	BLPP CONSTRAINT REGION OF EXAMPLE 4.1	49
FIGURE 4.3:	BRANCH AND BOUND TREE FOR EXAMPLE 4.1	42
FIGURE 5.1:	FLOW CHART OF THE BLPP ALGORITHM	56
FIGURE 5.2:	BLPP CONSTRAINT REGION OF EXAMPLE 5.1	64
FIGURE 5.3:	BRANCH AND BOUND TREE FOR EXAMPLE 5.1	67
FIGURE 5.4:	BRANCH AND BOUND TREE FOR EXAMPLE 5.2	73
FIGURE 6.1:	BLPP CONSTRAINT REGION OF EXAMPLE 6.1	92
FIGURE 6.2:	BLPP CONSTRAINT REGION OF EXAMPLE 6.2	94
FIGURE 6.3:	BRANCH AND BOUND TREE FOR EXAMPLE 6.2	96
FIGURE 6.4:	BLPP CONSTRAINT REGION OF EXAMPLE 6.3	111
FIGURE 6.5:	FLOW CHART OF MIXED INTEGER BLPP ALGORITHM	119
FIGURE 6.6:	BRANCH AND BOUND TREE FOR EXAMPLE 6.2	134
FIGURE B.1:	FEASIBLE REGIONS FOR EXAMPLE B.1	315

Chapter 1

INTRODUCTION

1.1 Introduction

The bilevel programming problem (BLPP) is a model for a leader-follower game in which two players are trying to maximize their individual objective functions, $F(x,y)$ and $f(x,y)$, respectively (see Basar and Selbuz 1979, or Simaan and Cruz 1973). Play is defined as sequential and the mood as uncooperative. The decision variables are partitioned amongst the players in such a way that neither can dominate the other. The leader goes first and through his choice of $x \in X \subseteq R^{n^1}$, is able to influence but not control the actions of the follower. This is achieved by reducing the set of feasible choices available to the follower. Subsequently, the follower reacts to the leader's decision by choosing a $y \in Y \subseteq R^{n^2}$ in an effort to

maximize his payoff. In so doing, he indirectly affects the leader's outcome.

The linear BLPP can be written in the following form.

$$\text{Max}_x F(x,y) = c^1x + c^2y \quad \text{where } y \text{ solves} \quad (1.1a)$$

$$\text{Max}_y f(x,y) = d^1x + d^2y \quad (1.1b)$$

$$\text{subject to} \quad Ax + By \leq b \quad (1.1c)$$

$$x \in X, y \in Y \quad (1.1d)$$

where c^1 and d^1 are n^1 -dimensional row vectors, c^2 and d^2 are n^2 -dimensional row vectors, A is an $m \times n^1$ -dimensional matrix, B is an $m \times n^2$ -dimensional matrix, b is an m -dimensional column vector, $X \subseteq R^{n^1}$, $Y \subseteq R^{n^2}$, and the nonnegativity requirements on the decision variables are subsumed in (1.1c). The purpose of this dissertation is to develop and test a set of algorithms to solve problem (1.1) and its mixed integer extensions.

1.2 Overview

The linear bilevel programming problem (BLPP) provides the focus of this dissertation. A review of

the work done on the BLPP is presented in Chapter 2. In Chapter 3, notation, definitions, and some theorems are given. The focus of Chapter 4 is the BLPP where all the variables are binary. An algorithm for solving this problem is created, and computational results are presented. In Chapter 5, an algorithm which utilizes branch and bound techniques to solve the continuous linear BLPP is developed. Computational experience is also reported. Chapter 6 centers on the mixed integer BLPP. The pitfalls encountered in working with this problem are discussed. Because of these pitfalls, algorithms which are heuristics are presented. These algorithms find good feasible solutions for the mixed integer case. The computational results achieved using these heuristics are shown. A production planning problem is examined in Chapter 7. Versions of this problem are presented which use the algorithm for the continuous linear BLPP and the algorithms for the mixed integer BLPP. Conclusions are presented in Chapter 8. In Appendix A, listings of the computer programs associated with the algorithms of Chapters 4, 5, and 6 are given. The more general multilevel programming problem is discussed in Appendix B.

CHAPTER 2

LITERATURE SEARCH

2.1 Introduction

Multiple criteria decision making has often been used to study the conflict that naturally arises in hierarchical systems (see Bard 1983c, Haimes et al. 1975, Mahoud 1977, Messarovic et al. 1970). A number of techniques that have proven successful in modeling the individual preferences and interactions which prevail in such systems include vector maximization (see Tarvainen and Haimes 1977, Yu and Zeleny 1981), multiattribute utility theory (see Keeney and Raiffa 1976, Zeleny 1982), and goal programming (see Charnes et al. 1967, Winkofsky et al. 1981). Their major drawback, however, stems from the need to postulate a single decision making unit with an accompanying set of goals for each group member. This formulation sidesteps the reactive nature of most multilevel organizations. In contrast,

game theoretic approaches explicitly consider the individual decision making units or players by assigning each a unique objective function and control set. When strategies are selected simultaneously, in a noncooperative manner, solutions are defined as equilibrium points (see Basar and Olsder 1979, Zangwill and Garcia 1981) so that at optimality no player can do better by unilaterally altering his choice. There are other types of noncooperative decision problems, though, where one (or more) of the players has the ability or authority to impose his preferences on a subset of the others. In such a situation, it is necessary to introduce a hierarchical solution concept known as a Stackelberg strategy (see Basar and Olsder 1982, Basar and Selbuz 1979, Vincent and Grantham 1981). The multilevel programming problem (MLPP) (see Bard 1982, Bard and Falk 1982a and 1982b, Basar and Olsder 1982, Bialas and Karwan 1982) conceptually extends the open-loop Stackelberg model to p players. Notions common to the latter; namely, the feasible region and the rational reaction set, are precisely defined for the general problem in Appendix B (also see Chang and Luh

1982a for a comparison with the inducible region); a working definition is given in Section 2.3.

2.2 Relevance of the Model

Multilevel programming can be relevant to almost any hierarchical system or organization in which independent agents make decisions in a sequential but interrelated fashion. Problems found in such areas as government regulation (see Bard 1983-1984, Cassidy et al. 1971), equipment scheduling (see Aoki and Satoh 1982), strategic warfare (see Bracken et al. 1977), and decentralized control (see Bard 1983, Burton and Obel 1977) readily lend themselves to this format. As an example, consider a firm concerned with product design and development. At the highest level there are overall corporate goals, such as market share, profit, and industrial leadership. At the next level there are research and design goals, and budgeting and technological constraints. At the bottom level, tradeoffs must be made between subsystem performance and design characteristics. The process is one of decision, reaction, and feedback, a familiar systems engineering problem.

In this example, if a lack of communications exists among the levels, or if lower divisions are competing against one another for company resources and producing diseconomies as a result; (e.g. see Bard 1983c, Baumol and Fabian 1964, Burton and Obel 1977), then multilevel programming should provide a suitable analytic framework (note that MLPP solutions are not necessarily Pareto-optimal, see Bard and Falk 1982b, Zeleny 1982). If, however, cooperation and central planning are the rule, another approach may be required.

2.3 Problem Formulation

Multilevel programming was first defined by Candler and Townsley (1982) as a generalization of mathematical programming. In this context, the constraint region is implicitly determined by a series of optimization problems which must be solved in a predetermined sequence.

Alternatively, the problem can be viewed as an n -person, nonzero-sum game with perfect information where the order of play is specified at the outset and the players' control sets are no longer assumed to be disjoint. As a consequence, the moves available to a player change as the game progresses and hence, may be

limited by the actions of the preceding players. When interdependent control sets are introduced, the difficulty of the overall problem markedly increases.

The problem that will be addressed differs from the conventional formulations of the n -person game in that the players are now required to move in turn. When the moves are assumed to occur simultaneously, disagreement often arises as to which of several measures is most likely to predict the actual outcome (e.g., see Luce and Raiffa 1957). To avoid such arguments, an appeal is made to the natural relationship that exists between the multilevel program and the standard mathematical program, and a solution is defined accordingly (Bard and Falk 1982a).

To formulate the problem, consider a system comprised of p levels, each characterized by individual objective functions f^i , $i=1, \dots, p$, defined over a jointly dependent constraint set S , which are to be maximized by the respective players. Assume that decisions are made sequentially beginning with player 1 who has control over a vector x^1 in X^1 , followed by player 2 who has control over a vector x^2 in X^2 , down through player p who has control over a vector x^p in X^p ,

where X^i ($i=1, \dots, p$) are nonempty subsets of R^{n^i} ; $X^i \cap X^j = \emptyset$, $i \neq j = 1, \dots, p$; $n = n^1 + \dots + n^p$; and $\underline{x} = (x^1, \dots, x^p) \in R^n$. Further assume that S is a compact subset of R^n , $\underline{x} \in S$, and each f^i maps S into R^1 . By implication, the choice made by a higher level player may affect the choices available to a lower level player through S ; the strategy selected by any member of the system, however, may influence the outcome realized by any other member through the latter's objective function. As Bialas and Karwan (1982) point out, this is in fact the nature of hierarchical systems where interacting decision making units execute their strategies in a preassigned order in an attempt to maximize net benefits. In addition, each player's control instruments may allow him to influence but not dictate the policies of another and thereby improve his own performance through the resultant externalities.

The following nested optimization problem known as the multilevel programming problem (MLPP) captures this structure:

$$\begin{array}{ll} \text{Max } f^1(x) & \text{where } x^2 \text{ solves} \\ x^1 \in X^1 & \end{array}$$

$$\begin{array}{ll} \text{Max } f^2(x) & \text{where } x^3 \text{ solves} \\ x^2 \in X^2 & \end{array}$$

.

.

(2.1)

.

$$\begin{array}{ll} \text{Max } f^{p-1}(x) & \text{where } x^p \text{ solves} \\ x^{p-1} \in X^{p-1} & \end{array}$$

$$\text{Max } f^p(x)$$

$$x^p \in X^p$$

subject to $\underline{x} \equiv (x^1, \dots, x^p) \in S$.

When $p = 1$, problem (2.1) reduces to a standard nonlinear program; when $p = 2$ and $f^1 = -f^2$ the general "max-min" problem (see Falk 1973) results. (A more formal definition of MLPP and related concepts is given in Appendix B.)

From an individual perspective, player p faces a problem parameterized in x^1 through x^{p-1} . Once these

variables are selected, he has only to solve a simple optimization problem. Player 1, however, is required to make the first move and therefore faces a problem implicit in x^2 through x^p . In effect, he must anticipate how each of the succeeding players will react to the control x^1 . Finally, player k faces a problem parameterized in x^1 through x^{k-1} and implicit in x^{k+1} through x^p . As shown in Appendix B, it is possible to inductively derive a feasible region for each player beginning at the lowest level which is level 1. An example for the linear three level programming problem is also given to highlight these concepts.

The principal difficulty in solving (2.1) stems from its nonconvexity, implying that direct solutions, even through dynamic programming are largely out of reach. A further difficulty occurs if multiple optimal solutions are present at one or more levels in the hierarchy. This may lead to the situation where not all the elements in a lower level player's rational reaction set (this set is defined in Appendix B) will actually provide the maximum payoff to some higher level player. In this case, one or more feasible regions, though perhaps bounded, will not be closed (e.g., see Bard and

Falk 1982a), so an exact solution may not be attainable. This will make it necessary to introduce the concept of ϵ -optimality (see Basar and Olsder 1982, Chang and Luh 1982b).

Throughout the dissertation a number of assumptions and additional notational conventions are adopted to simplify the presentation. First, the vector (x,y) and the function (F,f) are substituted for (x^1,x^2) and (f^1,f^2) when only two players are involved (that is, a single leader and follower). Next, it is assumed that X^i ($i=1,\dots,p$) are open subsets, so they are suppressed in the formulations. Finally, the set S is given the explicit form $\{\underline{x} : g(\underline{x}) \leq 0\}$ where $g(x)$ is an m -dimensional vector-valued function. Thus, the linear bilevel programming problem (BLPP) will be written as:

$$\text{Max}_x F(x,y) = c^1x + c^2y \text{ where } y \text{ solves} \quad (2.2a)$$

$$\text{Max}_y f(x,y) = d^1x + d^2y \quad (2.2b)$$

$$\text{subject to} \quad Ax + By \leq b \quad (2.2c)$$

$$x \geq 0, y \geq 0 \quad (2.2d)$$

where it should be noted that the term d^1x in the follower's objective function can be omitted without affecting the solution, and $g(\underline{x}) \leq 0$ is represented by

(2.2c) and (2.2d). In the next section relevant background material is presented along with current developments.

2.4 The Linear BLPP

The first advances toward a solution of the BLPP were made by Falk (1973) who studied the general linear max-min problem. His algorithm was based on branch and bound techniques, and required the solution of a number of linear programs at each iteration. Soon after, Gallo and Ulkucu (1977) devised a solution to the bilinear programming problem which is identical to Falk's formulation when the latter's constraint region is disjoint in the x and y spaces. While their algorithm was based on optimality conditions associated with the dual, others, including Konno (1976), have developed cutting plane approaches. More recently, Al-Khyyal and Falk (1983) have addressed the jointly constrained biconvex programming problem and have implemented an algorithm for some special cases.

Building on earlier work in parametric and nonconvex programming (e.g., see Gehner 1974, Vaish and Shetty 1976, Yu and Zeleny 1975), Bard (1984) has shown that the linear BLPP is equivalent to maximizing the

function F over a constraint region composed of connected edges and hyperplanes of the polyhedron S . This result confirmed the inherent nonconvexity of the linear MLPP and raised the possibility of local optima (see Bialas and Karwan 1982, Candler and Townsley 1982 as well). However, it was also shown that the global optimum occurred at a vertex of S , thereby suggesting an extreme point search procedure, perhaps within the framework of branch and bound.

Previously, Candler and Townsley (1982) proposed an implicit enumeration scheme that generated global information at each iteration. This information defined a set of necessary conditions that was then used to avoid returning to any previously explored bases. They observed that once an optimal basis to the follower's problem was obtained, changing the value of x might affect its feasibility but not its optimality. Therefore, through pivoting operations, it was possible to iteratively generate and test adjacent extreme points for feasibility and local optimality. Bialas and Karwan (1982) also pursued this idea while concurrently developing the " K^{th} -Best" Algorithm. The latter begins by generating an upper bound on F by solving

$$\text{Max } (F(x,y) : (x,y) \in S)$$

to obtain the "high point" (x',y') of S . This point is then checked for feasibility by solving the follower's problem with x fixed at x' . If the resultant value of y is equal to y' , the algorithm terminates with (x',y') as the global optimum; if not, all extreme points adjacent to (x',y') are checked for feasibility, and the one that produces the largest value for F is declared the solution so long as no other infeasible adjacent extreme point yields a larger payoff. If the test for optimality is not met, the next level of adjacent extreme points is checked and so on.

A second category of techniques for solving the BLPP is based on direct conversion of this problem into a standard nonconvex program. Working with the Kuhn-Tucker conditions of the follower's problem, Bard and Falk (1982a) developed the following equivalent representation:

$$\begin{array}{l} \text{Max } F(x,y) = c^1x + c^2y \\ x,y,u \end{array} \quad (2.3a)$$

$$\text{subject to} \quad uB = d^2 \quad (2.3b)$$

$$u(b - Ax - By) = 0 \quad (2.3c)$$

$$Ax + By \leq b \quad (2.3d)$$

$$x \geq 0, y \geq 0, u \geq 0 \quad (2.3e)$$

where $u \in R^m$. They solved this problem by converting (2.3c) into a piecewise linear, separable function and then applying a standard branch and bound algorithm (see Grotte 1976). The conversion requires the addition of the vector $z \in R^m$ to the decision space, ultimately increasing its order to $2m+n$. The dimensions of the problem that is finally solved, therefore, grows linearly as the number of constraints in S increases while the accompanying branch and bound tree grows exponentially.

Bialas, Karwan, and Shaw (1980) developed a second approach to (2.3) based on parametric complementary pivoting (see also Bialas and Karwan 1984). By adding the constraint $z = b - Ax - By$ to (2.3), where $z \in R^m$, and then eliminating (2.3c), they solve the resultant linear program with the further restriction that for each i , if u_i is in the basis then z_i must be excluded.

Formulation and operational requirements include the addition of approximately $3m$ variables to the original decision space, a parametric constraint of the form $c^1x + c^2y \geq t$, and the ability to store previously explored bases. Finally, Fortuny-Amat and McCarl (1981) solved (2.3) by converting the complementarity term (2.3c) into $2m$ mixed integer constraints with the addition of m zero-one variables.

2.5 The Linear MLPP

Work on the linear MLPP has progressed at a slightly slower pace. Bard (1982) developed the geometric properties for the three level programming problem (TLPP) and proposed a grid search algorithm that begins by setting up a parameterized linear program whose solution coincides with the linear version of (2.1). Although the approach works quite well for the BLPP (see Bard 1983b), it will not solve all problems and may further be limited by the bookkeeping burden imposed by the prospect of multiple optimal solutions.

A second approach offered by Wen (1981) combines the " K^{th} -Best" Algorithm to identify initial trial points, and the complementary pivot algorithm to test for bilevel optimality. Although this procedure seems

to work satisfactorily for most problems, its computational load grows geometrically with the number of constraints. In addition, the usefulness of the complementary pivot algorithm may be limited when degeneracy is present.

Bard and Falk (1982b) developed the rational reaction sets for each of the players while expanding on the geometric properties of the linear MLPP. Subsequently, first order necessary conditions were derived and the problem was recast as a standard nonlinear program. A cutting plane algorithm employing a vertex search procedure at each iteration was proposed to solve the linear 3-level case, but has not yet been fully automated. Limited testing suggests that problems with up to 100 variables and 100 constraints are well within the manageable range. An important consequence of this work was that individual sets of dual variables were identified for each level in the system. In accordance with their usual economic interpretation, it may be possible to develop a decomposition algorithm to solve large-scale problems.

2.6 The Nonlinear MLPP

To date, only a few specialized versions of the nonlinear MLPP have been addressed with any degree of success. Bard and Falk (1982a) investigated the "convex" BLPP where the follower's problem is a convex program. By assuming that f and $-g$ were both smooth and concave in y for x fixed, and that the regularity conditions held, they were able to reformulate the BLPP as a mathematical program:

$$\begin{array}{ll} \text{Max } F(x,y) & (2.4a) \\ x,y,u \end{array}$$

$$\text{subject to } \nabla_y f(x,y) - u \nabla_y g(x,y) = 0 \quad (2.4b)$$

$$ug(x,y) = 0 \quad (2.4c)$$

$$g(x,y) \leq 0 \quad (2.4d)$$

$$x \geq 0 \ y \geq 0, \ u \geq 0 \quad (2.4e)$$

Global solutions under the further assumption of function separability were obtained by using an existing routine. A second approach to the BLPP without regard to convexity was recently offered by Bard (1983a). The algorithm was based on a second set of necessary conditions developed by Bard (1984) for the leader's problem.

Other work on the multilevel problem has been primarily limited to variants of the Stackelberg game with strictly convex quadratic cost functions and linear equality constraints, in the context of optimal control (e.g., see Basar and Selbuz 1979, Tolwinski 1981). Luh, Chang, and Ning (1982) have also obtained some results for the discrete decision variable case, while Cruz (1978) has extended Stackelberg concepts to multilevel formulations.

Finally, a discussion of the the MLPP is related to bimatrix games (see Lemke and Howson 1964). Nonlinear programs with optimization problems in the constraints can be found in Bard and Falk (1982a).

CHAPTER 3

NOTATION, DEFINITIONS, AND THEOREMS

3.1 Introduction

Chapter 3 presents the notation which will be used in the remainder of the dissertation. Definitions of key concepts and theorems necessary to the development of the algorithms are also presented. Before presenting this information, the BLPP will be restated.

$$\underset{x}{\text{Max}} F(x,y) = c^1x + c^2y \quad \text{where } y \text{ solves} \quad (3.1a)$$

$$\underset{y}{\text{Max}} f(x,y) = d^1x + d^2y \quad (3.1b)$$

$$\text{subject to} \quad Ax + By \leq b \quad (3.1c)$$

$$x, y \geq 0 \quad (3.1d)$$

3.2 Notation

In the development it will be assumed that both the leader and follower have full knowledge of each other's objective function, and that no cooperation takes place.

This eliminates the use of side payments, and opens up the possibility of obtaining dominated solutions in the Pareto-optimal sense (Yu 1974).

Reference to the following notation will be made throughout the development.

BLPP Constraint Region:

$$\Omega = \{(x, y) \mid Ax + By \leq b\}$$

Follower's Feasible Region for $x \in X$ fixed:

$$\Omega(x) = \{y \in Y \mid Ax + By \leq b\}$$

Projection of Ω onto R^{n^1} :

$$\Gamma = \{x \in X \mid \exists y \in Y \text{ such that } Ax + By \leq b\}.$$

Follower's Rational Reaction Set:

$$\Psi(x) = \{y \mid y \text{ solves: } \max[f(x, y) : y \in \Omega(x)]\}$$

Inducible Region:

$$IR = \{(x, y) \mid x \in \Gamma, y \in \Psi(x)\}$$

3.3 Definitions

Using the above notation, two definitions can be given.

Definition 3.1: If $\bar{y} \in \Psi(\bar{x})$, then \bar{y} is said to be optimal with respect to \bar{x} . Such a pair as (\bar{x}, \bar{y}) will be called feasible.

Definition 3.2: A point (x^*, y^*) is said to be an optimal solution to the BLPP if

- (a) (x^*, y^*) is feasible;
- (b) for all feasible pairs $(\bar{x}, \bar{y}) \in \Omega$, $F(x^*, y^*) \geq F(\bar{x}, \bar{y})$.

In order to ensure that the problem is well defined, three additional requirements are imposed: 1) Ω is nonempty, 2) $\Psi(x^*)$ is a singleton for x^* optimal, and 3) for each decision taken by the leader, the follower has a feasible response; that is, $\Omega(x) \neq \emptyset$. The rational reaction set, $\Psi(x)$, denotes these responses, while the inducible region, IR , represents the set over which the leader may optimize. This implies that the BLPP can be written as

$$\text{Max}(F(x, y) : (x, y) \in IR) \quad (3.2).$$

3.4 Theorems

The following theorems are taken from Bard (1984), and they show that when the linear BLPP (3.1) is restated as a standard mathematical program in the form

of (3.2), the inducible region is comprised of connected faces of Ω .

Theorem 3.1: The linear BLPP (3.1) is equivalent to maximizing $c^1x + c^2y$ over a feasible region comprised of a piecewise linear equality constraint.

Because a linear function $c^1x + c^2y$ is being maximized over IR, and because $c^1x + c^2y$ is bounded above on Ω , the following can be concluded.

Corollary 3.1: The solution to the linear BLPP occurs at a vertex of the inducible region.

The fact that a vertex solution results was also postulated by Bialas and Karwan (1984) who noted that (3.1) could be written equivalently as

$$\text{Max}[c^1x + c^2y: (x,y) \in [IR]],$$

where $[IR]$ is the convex hull of IR. However, $[IR]$ is not equivalent to Ω , but in the next theorem, it is shown that the solution vertex of IR is also a vertex of Ω .

Theorem 3.2: The solution (x^*, y^*) of the linear BLPP occurs at an extreme point of Ω .

CHAPTER 4

AN ALGORITHM FOR SOLVING THE LINEAR 0-1 BILEVEL PROGRAMMING PROBLEM

4.1 The Bilevel Programming Problem with 0-1 Variables

The pure linear 0-1 bilevel programming problem (BLPP) takes the following form where $F(x,y)$ represents the leader's objective function, $f(x,y)$ represents the follower's objective function, x is an n^1 -dimensional vector which represents the binary variables controlled by the leader, and y is an n^2 -dimensional vector which represents the binary variables controlled by the follower. The problem is:

$$\underset{x}{\text{Max}} F(x,y) = c^1x + c^2y \quad \text{where } y \text{ solves} \quad (4.1a)$$

$$\underset{y}{\text{Max}} f(x,y) = d^1x + d^2y \quad (4.1b)$$

$$\text{subject to: } Ax + By \leq b \quad (4.1c)$$

$$x \in X = \{x_j \text{ binary} | j=1, \dots, n^1\} \quad (4.1d)$$

$$y \in Y = \{y_j \text{ binary} | j=1, \dots, n^2\} \quad (4.1e)$$

where c^1 , d^1 , c^2 , d^2 , A , B , and b are as defined in Section 1.1, and each is an array of integer constants. It should be noted that the term d^1x in the follower's objective function can be omitted without affecting the solution of the BLPP. This is true because once the leader has made his decision, the follower's problem becomes one in which the variables controlled by the leader are parameters.

4.2 The Parametric Formulation of the 0-1 BLPP

The BLPP with integer variables presented above cannot be solved using the techniques of integer programming. The key to solving (4.1) is in the recognition that any solution to (4.1) must be in the follower's rational reaction set, $\Psi(x)$. By limiting the search to this set while constantly seeking improvement in the leader's objective function, good points in the inducible region can be uncovered. This is achieved by formulating and iteratively solving the following parameterized integer program derived from (4.1).

$$\begin{array}{ll} \text{Max}_{x,y} f(y) = d^2 y & (4.2a) \end{array}$$

$$\text{subject to :} \quad Ax + By \leq b \quad (4.2b)$$

$$F(x,y) = c^1 x + c^2 y \geq \alpha \quad (4.2c)$$

$$\sum_{i=1}^{n^1} x_i \geq \beta \quad (4.2d)$$

$$x \in X, y \in Y \quad (4.2e)$$

where α and β are right-hand side parameters initially taken as $-\infty$ and 0, respectively. In problem (4.2), the follower's objective function with the $d^1 x$ term omitted is the objective function, and the leader's objective function becomes the parameterized constraint (4.2c). This constraint forces a tradeoff between the two objective functions (cf. Roodman 1972, Schrage and Wolsey 1985), while the parameterized constraint (4.2d) places a restriction on the sum of the variables controlled by the leader. Constraint (4.2d) is employed to prevent stalling in the iterative algorithm.

Proposition 4.1 defines when a point is in the inducible region (IR) or feasible region of problem (4.1).

Proposition 4.1: Let (\bar{x}, \bar{y}) solve problem (4.2). Then

$(\bar{x}, \bar{y}) \in \text{IR}$ if

$$c^1 \bar{x} + \sum_{j \in J^1} (\min[c_j^2, 0] - c_j^2) - \sum_{j \in J^0} (\max[c_j^2, 0] - c_j^2) \geq \alpha,$$

where $J^1 = \{j | \bar{y}_j = 1, j=1, \dots, n^2\}$ and

$J^0 = \{j | \bar{y}_j = 0, j=1, \dots, n^2\}$.

If (\bar{x}, \bar{y}) solves (4.2), one way to determine if (\bar{x}, \bar{y}) is in the IR of (4.1) is to fix x at \bar{x} and solve the resulting problem to determine if the solution is \bar{y} .

However, Proposition 4.1 defines a test which may be used to check if (\bar{x}, \bar{y}) is in the IR of (4.1). To apply Proposition 4.1, calculate the value of $c^1 \bar{x}$. Then adjust this value by subtracting $|c_j^2|$ for each j , $j=1, \dots, n^2$, such that $c_j^2 \geq 0$ and $y_j = 1$ or $c_j^2 \leq 0$ and $y_j = 0$.

Although Proposition 4.1 is valid when alternate optima exist, if $\Psi(x^*)$ is not a singleton as required, different values of y in $\Psi(x^*)$ may produce different objective function values, $F(x^*, y)$, for the leader. Basar and Selbuz (1979) address this ambiguity by

further defining what is meant by a solution of the game; alternatively, Bard and Falk (1982) propose solving a separate subproblem to test for the condition after their algorithm converges. Unfortunately, Proposition 4.1 did not provide a strong test for determining when a point was in the IR. Because of this, it is not included in the algorithm.

4.3 The Algorithm

The algorithm presented below applies a branch and bound procedure to find the solution of (4.1). The branch and bound scheme is a depth first approach centering on the leader's decision variables and, applied to problem (4.2), is used to find the optimal solution of (4.1). The basic idea is to successively examine points that satisfy the constraints of (4.2), fix the variables controlled by the leader at their corresponding values, and then resolve (4.2) to obtain a new point in the inducible region. By adjusting α and β at each iteration, the algorithm steadily increases the leader's objective function until the problem becomes infeasible.

A point, $(x^k, y^k) \in (\{0,1\} \times \{0,1\})^{n^1+n^2}$, is in the inducible region of the 0-1 BLPP when it satisfies two

conditions. The first condition is the constraint (4.2b), $Ax^k + By^k \leq b$, must be satisfied. The second condition is for x^k fixed, there does not exist a y^j such that $Ax^k + By^j \leq b$ and $f(x^k, y^j) > f(x^k, y^k)$.

Solving (4.2) readily yields feasible solutions of (4.1) when constraints (4.2c) and (4.2d) are relaxed and x is fixed. The algorithm exploits this property of (4.2).

Notationally, let $W = \{1, \dots, n^1\}$. Each path P_k in the branch and bound tree will correspond to an assignment of either $x_j = 0$ or $x_j = 1$ for $j \in W_k \subseteq W$, and thus identifies a partial solution for the variables controlled by the leader. The index set of the assigned variables controlled by the leader is denoted by W_k .

Now let

$$S_k^+ = \{j | j \in W_k \text{ and } x_j = 1\}$$

$$S_k^- = \{j | j \in W_k \text{ and } x_j = 0\}$$

$$S_k^0 = \{j | j \notin W_k\}$$

A completion of W_k is an assignment of binary values to the free variables controlled by the leader and is specified by the index set S_k^0 . The algorithm does not

specifically assign values to the variables controlled by the follower. These variables are always free and are assigned values which maximize the value of the follower's objective function in (4.2). A flow chart of the algorithm is presented in Figure 4.1.

Step 0: (Initialization) Put $k=0$, $S_k^+ = \phi$, $S_k^- = \phi$, $S_k^0 = \{1, \dots, n^1\}$. Put $\alpha = -\infty$, $\beta = 0$, and $\underline{F} = -\infty$.

Step 1: (Iteration k) Set $x_j = 1$ for $j \in S_k^+$ and $x_j = 0$ for $j \in S_k^-$. For the current values of α and β find a feasible solution of (4.2). If a feasible solution exists put $k \leftarrow k+1$, label the solution (x^k, y^k) and go to Step 2; otherwise go to Step 5.

Step 2: (Bounding) Fix x at x^k , α at $-\infty$, β at 0 and solve (4.2) to get a point (x^k, \hat{y}^k) . Compute $F(x^k, \hat{y}^k)$ and put $\underline{F} = \max[\underline{F}, F(x^k, \hat{y}^k)]$.

Step 3: (Branching) For each $x_j^k = 1$, $j=1, \dots, n^1$, such that $j \in S_k^0$, add a live vertex to the branch and bound tree, put $S_k^+ \leftarrow S_k^+ \cup \{j\}$, $S_k^0 \leftarrow S_k^0 \setminus \{j\}$, $S_k^- \leftarrow S_k^-$. The vertices are added to the branch and bound tree in a depth wise fashion.

Step 4: Let $\alpha = \underline{F} + 1$. Let $\beta = 1 + \sum_{j \in S_k^+} x_j$.
Go to Step 1.

Step 5: (Backtracking) If no live vertex exists, go to Step 6. Otherwise branch to the newest live vertex and update S_k^+ , S_k^- , and S_k^0 . Put $\beta=0$ and go to Step 1.

Step 6: (Termination) If $\underline{F} = -\infty$, there is no feasible solution to (4.1). Otherwise, declare the feasible point associated with \underline{F} the optimal solution.

Step 1 is designed to find a new point which is potentially bilevel feasible. At Step 1, it is not necessary to solve (4.2) to optimality. Step 1 requires

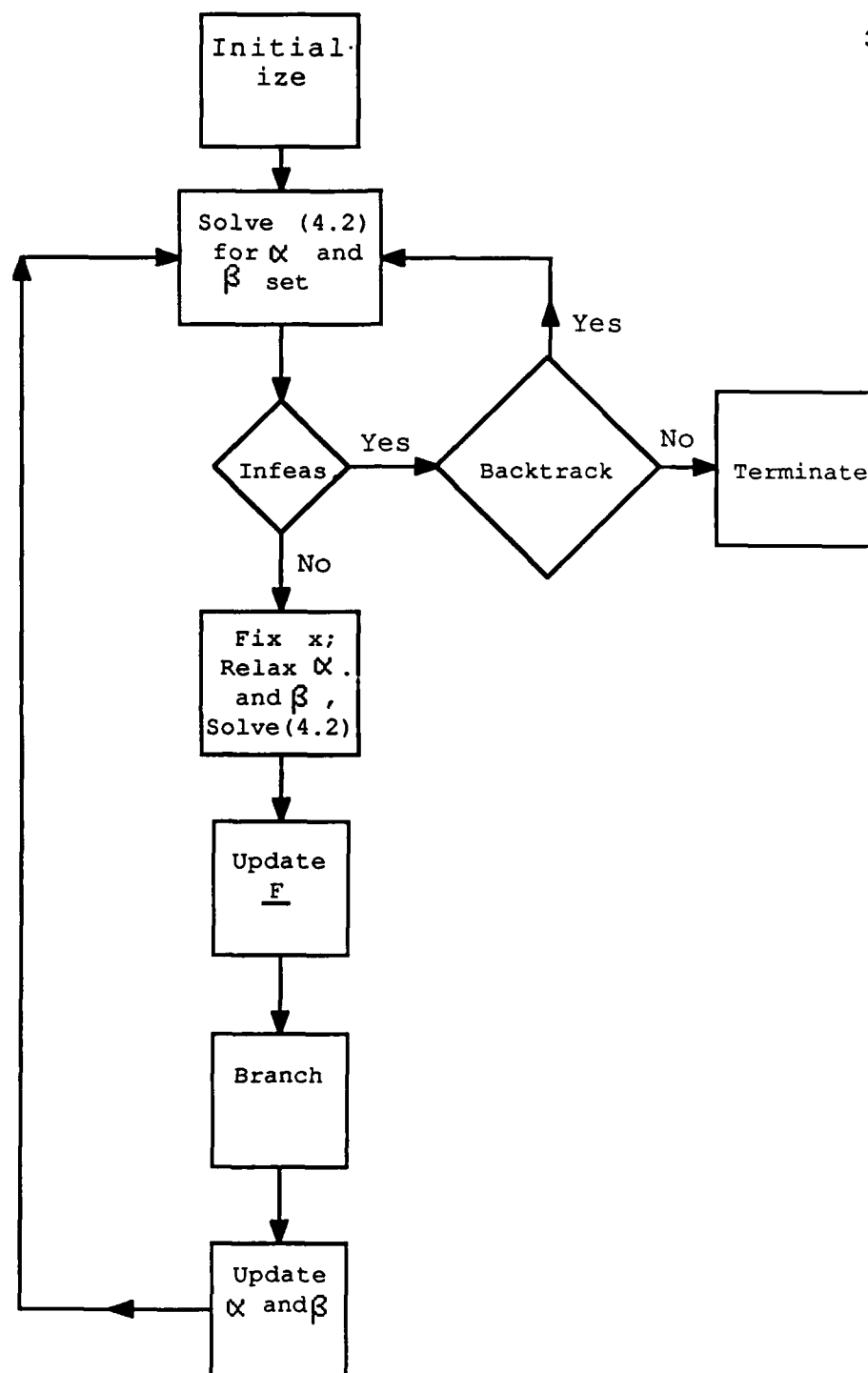


Figure 4.1: Flow Chart of 0-1 BLPP Algorithm

a feasible but not necessarily optimal solution of (4.2). However, if a feasible solution of (4.2) exists, it must be uncovered. At Step 2, constraints (4.2c) and (4.2d) are relaxed, and (4.2) is solved for a fixed vector of variables controlled by the leader. The solution is in the inducible region, and if an improvement results, the incumbent is replaced. (Alternatively, Proposition 4.1 could be invoked, but this would require solving (4.2) to optimality.) Branching occurs at Step 3 where S_k^+ , S_k^0 , and S_k^- are updated. At Step 3, live vertices are also added to the branch and bound tree. The live vertices are added in a depth wise manner. At Step 4, α is set to the current best known solution of (4.1) and is incremented by one. This guarantees that when the algorithm returns to Step 1, if a feasible solution, (x^k, y^k) , of (4.2) is found, it will satisfy $F(x^k, y^k) > \underline{F}$. Also at Step 4, β is set to the sum of the variables controlled by the leader and is incremented by one. This insures that at least one element of the set S_k^0 at iteration k has a value of 1, thus providing a branching variable at Step 3. If this operation is not included, the algorithm might return to the current point and hence stall.

Proposition 4.1 suggests that it may not be necessary to solve problem (4.2) at Step 2; however, a tradeoff exists. If the optimum is found, the need to solve (4.2) to optimality at Step 1 is obviated. Any feasible solution can be used at Step 2 to find a new point in the inducible region.

If problem (4.2) is infeasible at Step 1, the algorithm proceeds to Step 5 where the backtracking operation is performed. To facilitate bookkeeping, the path P_k in the branch and bound tree can be concisely represented by an ℓ -dimensional vector where ℓ is the current depth of the tree. The order of the components is determined by their "level" in the tree. Indices only appear in the vector P_k if they are in S_k^+ or S_k^- and appear underlined if they are in S_k^- . Because the algorithm always branches to 1 first, the backtracking operation may be accomplished by finding the rightmost nonunderlined element of P_k , underlining it, and erasing all entries to the right. The newly underlined entry is deleted from S_k^+ and added to S_k^- ; the erased entries are deleted from S_k^- and added to S_k^0 . If this procedure is followed, the optimal solution of the BLPP cannot be overlooked. This leads to the following result.

Proposition 4.2: Under the uniqueness assumption stated in Chapter 3, the algorithm terminates with the optimal solution of the BLPP (4.1).

Proof - The algorithm implicitly looks at all combinations of x at Step 3 and 5; the subproblem solved at Step 2 guarantees that all incumbent solutions are in the inducible region. ■

4.4 Examples

An example is taken from Bard (1984) and is used to demonstrate how the algorithm works. Coincidentally, it illustrates two interesting features of the solution of the integer linear BLPP.

Example 4.1

Max $F(x,y) = -x - y$, where y solves
 $x \geq 0$

Max $f(x,y) = 5x + y$
 $y \geq 0$

subject to

$$-x - y/2 \leq -2$$

$$-x/4 + y \leq 2$$

$$x + y/2 \leq 8$$

$$x - 2y \leq 4$$

The optimal solution of Example 4.1 without regard to integrality is $(x^*, y^*) = (8/9, 20/9)$ with $F = -28/9$. The graph of the feasible region, Ω , is shown in Figure 4.2, and it indicates that $x < 8$ and $y < 4$. If x and y are restricted to integer values, the example can be reformulated as a linear BLPP with 0-1 variables where $x = x_1 + 2x_2 + 4x_3$ and $y = y_1 + 2y_2$. The example with binary variables is restated as Example 4.2.

Example 4.2

Max $F(x, y) = -x_1 - 2x_2 - 4x_3 - y_1 - 2y_2$, where y solves
 x

$$\text{Max}_y f(x, y) = 5x_1 + 10x_2 + 20x_3 + y_1 + 2y_2$$

subject to

$$-2x_1 - 4x_2 - 8x_3 - y_1 - 2y_2 \leq -4$$

$$-x_1 - 2x_2 - 4x_3 + 4y_1 + 8y_2 \leq 8$$

$$2x_1 + 4x_2 + 8x_3 + y_1 + 2y_2 \leq 16$$

$$x_1 + 2x_2 + 4x_3 - 2y_1 - 4y_2 \leq 4$$

$$x_1, x_2, x_3, y_1, y_2 = 0 \text{ or } 1$$

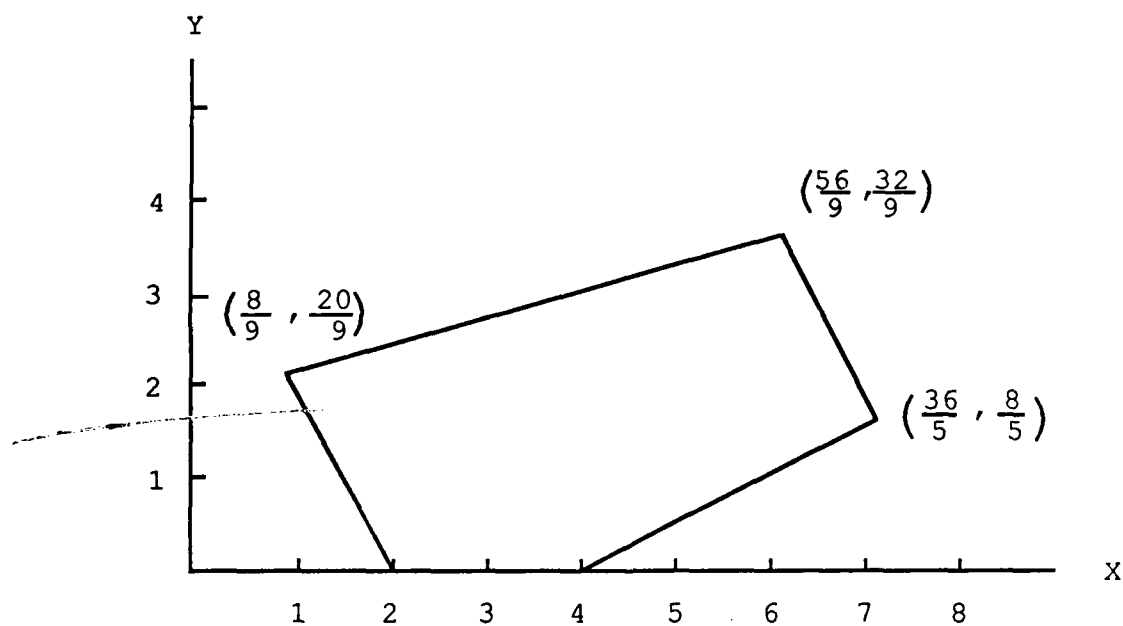


Figure 4.2: BLPP Constraint Region of Example 4.1

Example 4.2 can now be restated in the form of (4.2).

$$\begin{array}{l} \text{Max} \quad y_1 + 2y_2 \\ \text{subject to} \end{array}$$

$$-2x_1 - 4x_2 - 8x_3 - y_1 - 2y_2 \leq -4$$

$$-x_1 - 2x_2 - 4x_3 + 4y_1 + 8y_2 \leq 8$$

$$2x_1 + 4x_2 + 8x_3 + y_1 + 2y_2 \leq 16$$

$$x_1 + 2x_2 + 4x_3 - 2y_1 - 4y_2 \leq 4$$

$$F(x,y) = -x_1 - 2x_2 - 4x_3 - y_1 - 2y_2 \geq \alpha$$

$$x_1 + x_2 + x_3 \geq \beta$$

$$x_1, x_2, x_3, y_1, y_2 = 0 \text{ or } 1$$

The first few steps of the algorithm are outlined below. For the sake of brevity, the integer representation of the binary points is presented.

After initialization at Step 0, the algorithm finds the point (4,3) at Step 1 which is confirmed to be in the inducible region at Step 2. Thus, $\underline{F} = -7$, $S_1^+ = \{3\}$, $S_1^0 = \{1,2\}$, $\alpha = -6$, and $\beta = 2$. Solving again at Step 1 yields the point (5,1). At Step 2, $(x^2, \hat{y}^2) = (5,3)$ and

$F(\hat{x}^2, \hat{y}^2) = -8$, so \underline{F} remains -7 . Updating gives $S_2^+ = \{3, 1\}$, $S_2^0 = \{2\}$, $\alpha = -6$, and $\beta = 3$. At Step 1, no feasible solution is found, so the algorithm goes to Step 5 and backtracks giving $S_2^+ = \{3\}$, $S_2^- = \{1\}$, $S_2^0 = \{2\}$, $\alpha = -6$, and $\beta = 0$. Returning to Step 1, the calculations continue until convergence occurs at the 11th iteration.

The optimal solution of Example 4.2 is $(x^*, y^*) = (1, 2)$ with $F = -3$. The corresponding branch and bound tree is shown in Figure 4.3. During execution, the algorithm returns to Step 1 thirteen times in an attempt to improve upon the incumbent. Subsequently, seven subproblems are solved at Step 2 to find new points in the inducible region. If problem (4.2) were completely solved at Step 1, however, it would not have been necessary to solve three of these seven subproblems.

With regard to individual outcomes, note that the leader does better when the variables are restricted to integer, rather than continuous, values. Although this observation cannot be generalized, it arises here because the follower's feasible region $\Omega(x)$ is somewhat reduced by the integer requirements. When the first

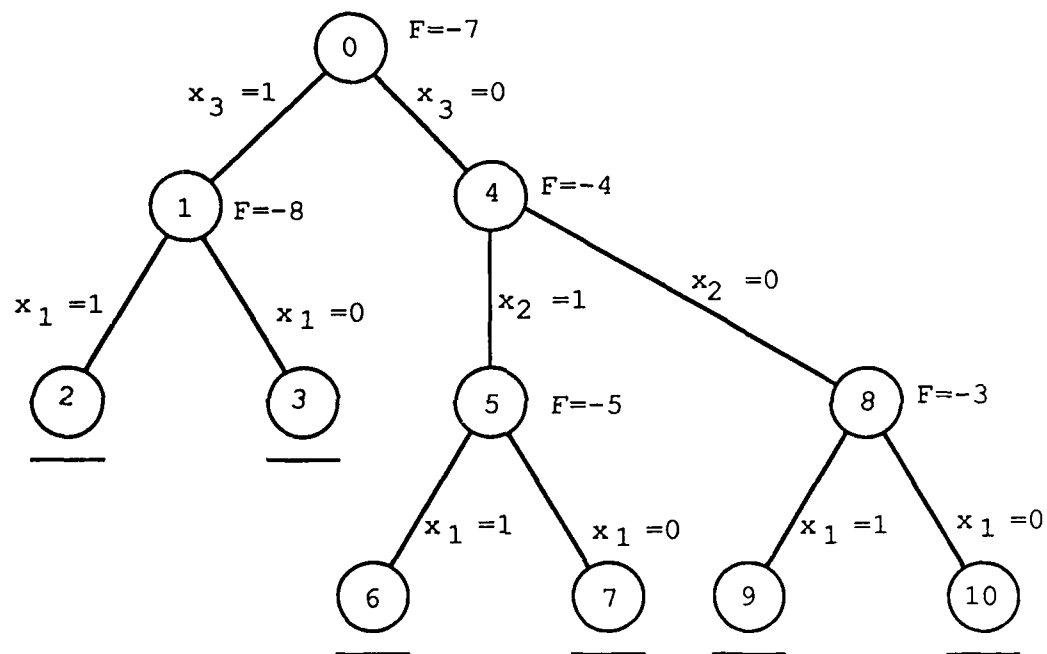


Figure 4.3: Branch and Bound Tree for Example 4.1

term in $f(x,y)$ is omitted, the opposite is true for the follower.

A second inference that can be drawn from the example concerns the nature of the solution. In particular, note that when compared to the optimum $(1,2)$, the point $(2,0)$ provides better outcomes for both players but is not in the inducible region. That is, if the leader plays $x=2$, the follower chooses $y=2$, thereby degrading the leader's payoff by 1 unit. Thus, solutions to the integer BLPP are not necessarily Pareto-optimal. (If the first term in $f(x,y)$ is dropped, a slight alteration in the problem would be required to demonstrate this result. Although the first term in $f(x,y)$ does not affect the solution of the BLPP, it does affect the value of the follower's objective function.)

4.5 Computational Experience

A variety of randomly generated problems are solved using the algorithm. The problems are generated using a random number generator based on Knuth (1969). In the randomly generated problems, all constraints are inequalities with positive right-hand sides which vary between 0 and 45. The coefficients of the A- and B-

matrices range from -10 to 28 with approximately 40 percent of the nonzero coefficients negative. The coefficients of the leader's and follower's objective functions also assume both positive and negative values. The number of constraints in each problem was set to 0.4 times the number of variables in the problem, and the density of the A- and B-matrices is approximately 0.475.

The algorithm is written in FORTRAN 77 on an IBM 3081D. A Balas-type algorithm similar to the one described by Garfinkel and Nemhauser (1972) is used to solve the 0-1 problem at Steps 1 and 2. In their presentation, Garfinkel and Nemhauser suggest several tests which may reduce the enumeration at the expense of additional calculation. The only test included in the algorithm is Test 1 which checks for feasibility of a completion. Tests 2-4 were evaluated but failed to provide an improvement in performance. The FORTRAN code of the additive algorithm is based on a code written by Jensen (1983).

In all, 10 problems were run for each case where a case is defined by the total number of variables and the number of variables controlled by the follower. The information presented in Table 4.1 is the average for

each case. CPU time is given in seconds. 'Iterations-Step 1' refers to the number of times Step 1 of the algorithm is performed and represents the number of times the algorithm searches for a feasible solution of (4.2). In a similar fashion, 'Iterations-Step 2' refers to the number of times the algorithm attempts to find an improved feasible solution of the BLPP at Step 2. 'No. of Verts' indicates the number of vertices in the branch and bound tree created by the algorithm. 'Optimal Solution (Vert)' is the vertex in the branch and bound tree at which the optimal solution of the BLPP is uncovered.

As expected, the CPU time grows exponentially with the size of the problem but seems to be independent of the way the variables are partitioned. Nevertheless, the averages given in Table 4.1 do not reveal the large differences that were observed among problems of equivalent size. As an example, the greatest range in CPU time appeared for those problems containing 45 variables where 15 are controlled by the follower. Here, the longest CPU time is 757 seconds while the shortest is less than 3 seconds. Eliminating these two extremes and redoing the calculations gives an average

TABLE 4.1: COMPUTATIONAL RESULTS

No. of Variables ($n^1 + n^2$)	Follower Variables (n^2)	CPU Time (sec)	Iterations Step 1	Step 2	No. of Verts	Optimal Solution (Vert)
15	5	0.06	26	14	23	13
20	5	0.10	24	11	25	15
20	10	0.37	47	29	34	18
25	5	0.51	39	18	39	26
25	10	1.87	141	80	119	50
30	5	0.98	47	19	53	44
30	10	1.23	44	21	46	24
30	15	3.11	47	25	43	26
35	5	5.10	36	14	42	31
35	10	4.97	49	23	50	41
35	15	13.49	86	49	71	54
40	5	14.06	47	15	61	47
40	10	31.87	52	22	58	45
40	15	18.58	47	21	51	31
40	20	21.78	44	21	44	28
45	10	23.08	20	8	22	16
45	15	147.66	50	20	57	48
45	20	106.86	124	64	117	106

of 89.6 seconds for the remaining eight problems.

Similar observations can be made concerning the number of times it was necessary to execute Steps 1 and 2 within the same problem set. The greatest variation occurred in the case with 25 variables where 10 variables are controlled by the follower. For one problem, the algorithm performed Step 1 219 times and Step 2 125 times. For another problem of this size, the algorithm

performed Step 1 seven times and Step 2 three times. The averages with these two extremes eliminated is Step 1-32 and Step 2-12. For problems with 50 or more variables (results not shown in Table 4.1), computation times often exceeded 900 seconds, the cutoff time chosen for the runs.

After solving over 200 randomly generated problems, it became clear the performance of the algorithm is highly problem dependent. The cases cited above highlight the wide variation encountered in solving problems of the same size.

CHAPTER 5

A BRANCH AND BOUND ALGORITHM FOR THE BILEVEL PROGRAMMING PROBLEM

5.1 Development of the Kuhn-Tucker Approach

In this chapter, the focus is on the linear bilevel programming problem (BLPP) with continuous variables. In the following discussion let $F(x,y)$ represent the objective function of the leader and let $f(x,y)$ represent the objective function of the follower where $x \in X \subseteq R^{n^1}$ is the n^1 -dimensional vector of decision variables controlled by the leader and $y \in Y \subseteq R^{n^2}$ is the n^2 -dimensional vector of decision variables controlled by the follower. The linear BLPP is now formulated with explicit nonnegativity constraints on the variables.

$$\text{Max}_x F(x,y) = c^1x + c^2y \text{ where } y \text{ solves} \quad (5.1a)$$

$$\text{Max}_y f(x,y) = d^2y \quad (5.1b)$$

$$\text{subject to} \quad Ax + By \leq b \quad (5.1c)$$

$$x \geq 0, y \geq 0 \quad (5.1d)$$

where c^1 , c^2 , d^2 , A , B , and b are as defined in Section 1.1.

When the leader has selected values for the decision variables he controls, the follower's problem becomes a standard linear programming problem parameterized in x . Because the follower's problem is a linear programming problem parameterized in x , its solution must satisfy the Kuhn-Tucker conditions of the follower's problem. The follower's problem derived from (5.1) when x is fixed is

$$\text{Max}_y f(y) = d^2y \quad (5.2a)$$

$$\text{subject to} \quad By \leq b - Ax \quad (5.2b)$$

$$y \geq 0 \quad (5.2c)$$

The Kuhn-Tucker conditions of (5.2) are

$$By \leq b - Ax \quad (5.3a)$$

$$d^2 - u^1 B + u^2 I = 0 \quad (5.3b)$$

$$u^1 (b - Ax - By) = 0 \quad (5.3c)$$

$$u^2 y = 0 \quad (5.3d)$$

$$u^1, u^2, y \geq 0 \quad (5.3e)$$

where x is fixed, u^1 is an m -dimensional row vector of Kuhn-Tucker multipliers, u^2 is an n^2 -dimensional row vector of Kuhn-Tucker multipliers associated with the nonnegativity constraints on the variables controlled by the follower, and I is an $n^2 \times n^2$ identity matrix. For ease of presentation, the set of functions $g(x,y)$ is introduced. For $i=1, \dots, m$, $g_i(x,y) = b - Ax - By$, and for $i=m+1, \dots, m+n^2$, $g_i(x,y) = y$. Also, define $u \equiv (u^1, u^2)$ where u is an $m+n^2$ -dimensional row vector of Kuhn-Tucker multipliers associated with the constraint $g(x,y) \geq 0$. Using this notation, (5.1) can be rewritten

$$\text{Max}_x F(x,y) \quad \text{where } y \text{ solves} \quad (5.4a)$$

$$\text{Max}_y f(x,y) \quad (5.4b)$$

$$\text{subject to} \quad g(x,y) \geq 0 \quad (5.4c)$$

$$x \geq 0 \quad (5.4d)$$

In the above formulation, the follower's problem can be replaced by its Kuhn-Tucker conditions (Bard and Falk 1982, Bialas and Karwan 1984, Fortuny-Amat and McCarl 1981). The result is the following equivalent representation of the BLPP (5.1).

$$\begin{array}{ll} \text{Max } F(x,y) & (5.5a) \\ x,y,u \end{array}$$

$$\text{subject to } g(x,y) \geq 0 \quad (5.5b)$$

$$\nabla_y[f(x,y)] + u \cdot \nabla_y[g(x,y)] = 0 \quad (5.5c)$$

$$u \cdot g(x,y) = 0 \quad (5.5d)$$

$$u, x \geq 0 \quad (5.5e)$$

where ∇ represents the gradient operator, and ∇_y is the gradient with respect to y . Bard and Falk (1982) solve problem (5.5) using a nonconvex programming algorithm based on a branch and bound approach. Fortuny-Amat and McCarl (1981) force satisfaction of the complementary slackness conditions by transforming (5.5) into a larger mixed integer programming problem. Bialas and Karwan (1984) use a Parametric Complementary Pivot algorithm to solve (5.5). Their algorithm is limited to cases where c^2 , the coefficients in the leader's objective function of the variables controlled by the follower, must be

nonnegative. The Parametric Complementary Pivot algorithm produces a solution which is within $\gamma \times 100\%$ of the optimal objective function value where γ is a small positive scalar. Also, Bialas and Karwan's (1984) algorithm does not handle the degenerate case. Theorem 3.2 states the solution of (5.1) occurs at an extreme point of the constraint region defined by (5.1c)-(5.1d). Using this knowledge, Bialas and Karwan (1984) develop an algorithm which finds the optimal solution of (5.1) by testing extreme points of the BLPP constraint region. Another way to solve this problem is a branch and bound procedure which forces satisfaction of the complementary slackness conditions.

5.2 The Algorithm

Using a branch and bound approach applied to (5.5), the following algorithm finds the solution of (5.1). The problem defined by (5.5a-5.5c, 5.5e) is a linear program which can be solved using standard techniques. However, a solution of the BLPP must also satisfy (5.5d). The basic idea of the algorithm is to suppress the complementarity term (5.5d) and solve the resulting linear subproblem. At each iteration of the algorithm, a check is made to determine if the complementary

slackness conditions are satisfied. If so, the corresponding point is in the inducible region, and hence, a potential solution to (5.1); if not, a branch and bound scheme is used to implicitly examine all combinations of complementary slackness conditions.

Thus, the algorithm solves the following linear programming problem and forces satisfaction of the complementary slackness conditions by use a branch and bound procedure.

$$\begin{array}{ll} \text{Max } F(x,y) & (5.6a) \\ x,y,u \end{array}$$

$$\text{subject to } g(x,y) \geq 0 \quad (5.6b)$$

$$\nabla_y[f(x,y)] + u \cdot \nabla_y[g(x,y)] = 0 \quad (5.6c)$$

$$x, u \geq 0 \quad (5.6d)$$

Before presenting the algorithm, the notation used in the branch and bound process is defined. Similar to Chapter 4, let $W = \{1, \dots, m, m+1, \dots, m+n^2\}$ be the index set of the complementary slackness conditions of (5.5d), and let \underline{F} be the incumbent lower bound on the leader's objective function. Each path P_k in the branch and bound tree corresponds to an assignment of either $u_i = 0$ or $g_i = 0$ for $i \in W_k \subseteq W$. Now let

$$S_k^+ = \{i \mid i \in W_k \text{ and } u_i = 0\}$$

$$S_k^- = \{i \mid i \in W_k \text{ and } g_i = 0\}$$

$$S_k^0 = \{i \mid i \notin W_k\}.$$

For $i \in S_k^0$, the variables u_i and g_i are free to assume any nonnegative values in the solution of (5.5) with (5.5d) suppressed, so (5.5d) will not necessarily be satisfied. The algorithm's flow chart is shown in Figure 5.1.

Step 0: (Initialization) Put $k = 0$, $S_k^+ = \phi$, $S_k^- = \phi$, $S_k^0 = \{1, \dots, m+n^2\}$, and $\underline{F} = -\infty$.

Step 1: (Iteration k) Set $u_i = 0$ for $i \in S_k^+$ and $g_i = 0$ for $i \in S_k^-$. Attempt to solve (5.6). If the resultant problem is infeasible, go to Step 5; otherwise put $k \leftarrow k+1$ and label the solution (x^k, y^k, u^k) .

Step 2: (Fathoming) If $F(x^k, y^k) \leq \underline{F}$, go to Step 5.

Step 3: (Branching) If $u_i^k \cdot g_i(x^k, y^k) = 0$, $i=1, \dots, m+n^2$, go to Step 4; otherwise, select i for which $u_i^k \cdot g_i(x^k, y^k)$ is largest and label it i_1 . Put $S_k^+ \leftarrow S_k^+ \cup \{i_1\}$, $S_k^0 \leftarrow S_k^0 \setminus \{i_1\}$, $S_k^- \leftarrow S_k^-$, add a live vertex to the branch and bound tree, and go to Step 1.

Step 4: (Updating) $\underline{F} = F(x^k, y^k)$.

Step 5: (Backtracking) If no live vertex exists, go to Step 6. Otherwise branch to the newest live vertex and update S_k^+ , S_k^- , and S_k^0 . Go to Step 1.

Step 6: (Termination) If $\underline{F} = -\infty$, there is no feasible solution to (5.1). Otherwise, declare the feasible point associated with \underline{F} the optimal solution.

Step 1 is designed to find a new point which is potentially bilevel feasible. If no solution exists, or the solution does not offer an improvement over the incumbent (Step 2), the algorithm goes to Step 5 and

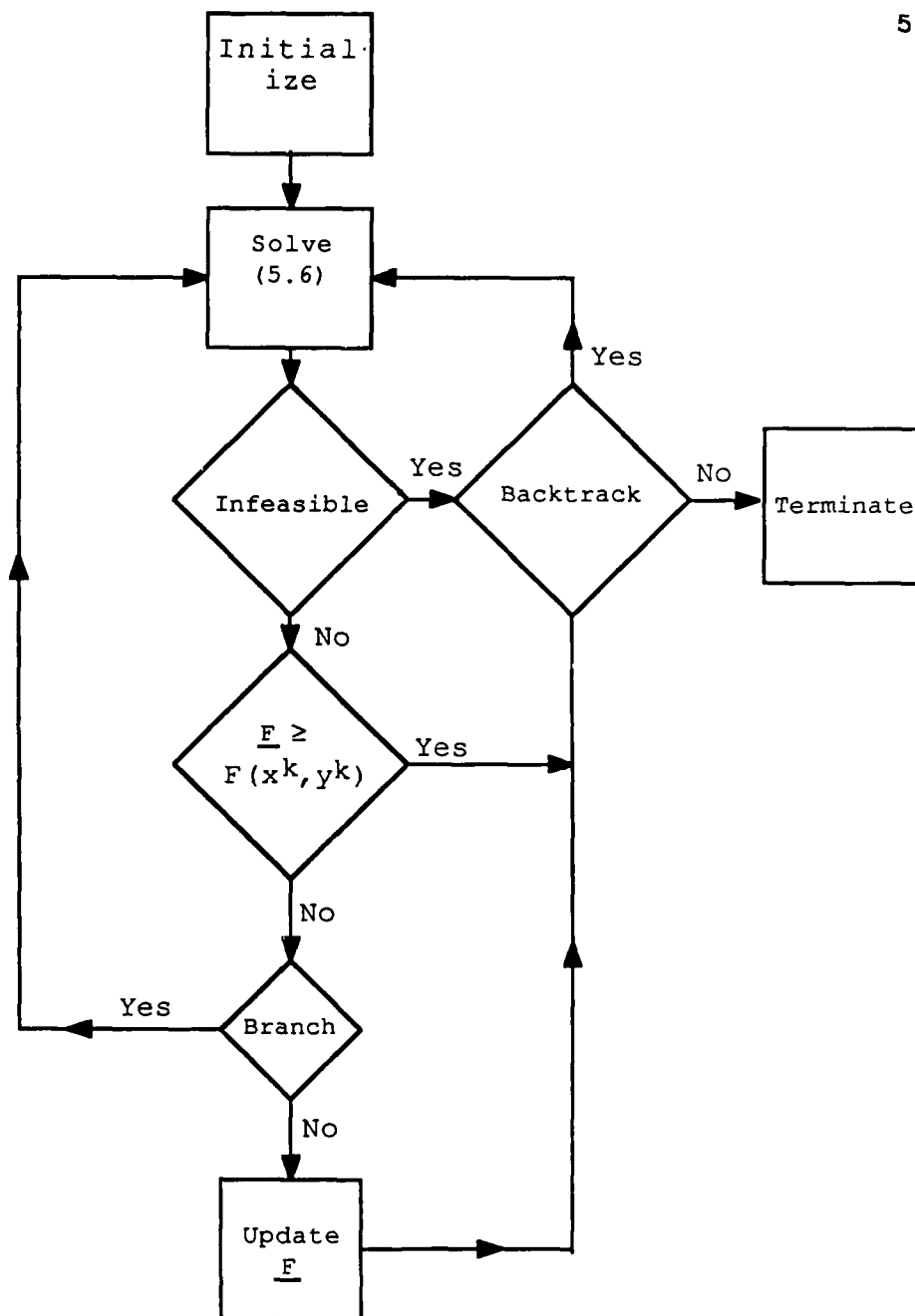


Figure 5.1: Flow Chart of the BLPP Algorithm

backtracks. At Step 3, a check is made to determine if the complementary slackness conditions are satisfied. Confirmation indicates that a feasible solution of the bilevel program has been found, and at Step 4, the lower bound on the leader's objective function is updated. Alternatively, if the complementary slackness conditions are not satisfied, the term with the largest product is used at Step 3 to provide the branching variable. Branching is always done on the Kuhn-Tucker multiplier, and vertices are added to the tree in a depth wise manner.

At Step 5, the backtracking operation is performed. Once again, to facilitate bookkeeping, the path P_k in the branch and bound tree is represented by an ℓ -dimensional vector where ℓ is the current depth of the tree. The order of the components of P_k is determined by their "level" in the tree. Indices only appear in P_k if they are in either S_k^+ or S_k^- with the entry underlined if they are in S_k^- . Because the algorithm always branches on a Kuhn-Tucker multiplier first, backtracking is accomplished by finding the rightmost nonunderlined element of P_k , underlining it, and erasing all entries

to the right. The newly underlined entry is deleted from S_k^+ and added to S_k^- ; the erased entries are deleted from S_k^- and added to S_k^0 . This leads to the following result.

Proposition 5.1: Under the uniqueness assumption associated with the rational reaction set, $\Psi(x)$, the algorithm terminates with the optimal solution of the BLPP (5.1).

Proof: The algorithm forces satisfaction of the complementary slackness conditions of (5.5) which is an equivalent representation of (5.1). By implicitly considering all combinations of $u \cdot g(x,y)$ at Steps 3 and 5, the optimal solution cannot be overlooked. ■

5.3 Other Forms of the BLPP

The BLPPs which the algorithm solves are not limited to those which take the form of (5.1). Because the follower's problem is converted into an equivalent form using the Kuhn-Tucker conditions, the linearity of the equivalent problem (5.5) is maintained when the follower's objective function, $f(x,y)$, is quadratic:

$f(x,y) = d^2y + \frac{1}{2}(x,y)^T Q(x,y)$ where Q is a symmetric matrix whose lower $n^2 \times n^2$ submatrix is negative semidefinite and T is the transpose operator. Equality constraints may also be included in the BLPP. The Kuhn-Tucker multipliers associated with these constraints are unrestricted in value, so they would not be included in the complementary slackness conditions. The algorithm can also solve the BLPP where some or all of the decision variables have upper (and lower) bounds. Thus, the BLPP which the algorithm solves has the following form

$$\text{Max}_x F(x,y) = c^1x + c^2y \text{ where } y \text{ solves} \quad (5.7a)$$

$$\text{Max}_y f(x,y) = d^2y + \frac{1}{2}(x,y)^T Q(x,y) \quad (5.7b)$$

$$\text{subject to} \quad Ax + By \leq b^1 \quad (5.7c)$$

$$Cx + Dy = b^2 \quad (5.7d)$$

$$0 \leq x \leq U^1 \quad (5.7e)$$

$$0 \leq y \leq U^2 \quad (5.7f)$$

where c^1 , c^2 , and d^2 are as defined in Section 1.1, A is an $m^1 \times n^1$ dimensional matrix, B is an $m^1 \times n^2$ dimensional matrix, C is an $m^2 \times n^1$ dimensional matrix, D is an $m^2 \times n^2$ dimensional matrix, b^1 is an m^1 -dimensional column vector, b^2 is an m^2 -dimensional

column vector, U^1 is an n^1 -dimensional column vector, U^2 is an n^2 -dimensional column vector, T is the transpose operator, and Q is a symmetric $(n^1+n^2) \times (n^1+n^2)$ -matrix whose lower $n^2 \times n^2$ submatrix is negative semidefinite. In coding the algorithm, a lower triangular $(n^1+n^2) \times (n^1+n^2)$ -matrix, L , is used to ease data input requirements. The matrix L is used in the following discussion to aid presentation of the Kuhn-Tucker representation of (5.7). Since L is a lower triangular matrix, the quadratic portion of (5.7b) can be represented by the entry of the coefficients of the nonlinear terms with one entry per term. Before using the Kuhn-Tucker approach to write the equivalent representation of (5.7), a more detailed picture of the L matrix is given. The L matrix may be partitioned as follows

$$L = \left[\begin{array}{c|c} L_1 & 0 \\ \hline L_2 & L_3 \end{array} \right]$$

where L_1 is an $n^1 \times n^1$ lower triangular matrix, L_2 is an $n^2 \times n^1$ matrix, and L_3 is an $n^2 \times n^2$ lower triangular

matrix. Also, define the diagonal of L_3 as the n^2 -dimensional vector ℓ . Derive a symmetric matrix with zeros down the diagonal from L_3 . Derive this matrix by setting the entries on the diagonal of L^3 to zero. Then add this matrix to its transpose. The resulting symmetric matrix is designated L_3^s . This manipulation of L eases formulation of the Kuhn-Tucker representation of (5.7).

The Kuhn-Tucker equivalent of (5.7) is

$$\begin{aligned} \text{Max } F(x, y) &= c^1x + c^2y \\ x, y, u^1, u^2, u^3, u^4 \end{aligned} \quad (5.8a)$$

$$\text{subject to} \quad Ax + By \leq b^1 \quad (5.8b)$$

$$Cx + Dy = b^2 \quad (5.8c)$$

$$0 \leq x \leq U^1 \quad (5.8d)$$

$$0 \leq y \leq U^2 \quad (5.8e)$$

$$d^2 + L_2x + L_3^sy + 2\ell y - u^1B + u^2D + u^3 - u^4 = 0 \quad (5.8f)$$

$$u^1(b^1 - Ax - By) = 0 \quad (5.8g)$$

$$u^3y = 0 \quad (5.8h)$$

$$u^4(U^2 - y) = 0 \quad (5.8i)$$

$$u^1, u^3, u^4 \geq 0 \quad (5.8j)$$

$$u^2 \text{ unrestricted} \quad (5.8k)$$

where u^1 , u^2 , u^3 , and u^4 are vectors of Kuhn-Tucker multipliers with dimension m^1 , m^2 , n^2 , and n^2 , respectively, and are associated with constraint (5.8b), constraint (5.8c), the nonnegative restriction on y , and the upper bound constraints on y , respectively. Equation (5.8f) represents the linear set of constraints derived when the gradient is taken with respect to y .

5.4 Examples

Three examples are presented to illustrate the algorithm. The first is taken from Bard (1984). This example is the same one used to illustrate the 0-1 BLPP algorithm in Chapter 4. The second is taken from Candler and Townsley (1982) and serves to contrast the performance of the various iterative schemes. The third example contains nonlinear terms in the follower's objective function and also has an equality constraint and an upper bounded variable.

Example 5.1

$$\begin{array}{l} \text{Max } F(x,y) = -x -y \text{ where } y \text{ solves} \\ x \end{array}$$

$$\begin{array}{l} \text{Max } f(x,y) = 5x + y \\ y \end{array}$$

subject to

$$x + y/2 \geq 2$$

$$\begin{aligned}
 -x/4 + y &\leq 2 \\
 x + y/2 &\leq 8 \\
 x - 2y &\leq 4 \\
 x \geq 0, y &\geq 0
 \end{aligned}$$

The graph of Example 5.1 is shown in Figure 5.2, and its equivalent representation using the Kuhn-Tucker approach is

$$\begin{aligned}
 \text{Max } F(x,y) &= -x - y \\
 x, y, u
 \end{aligned}$$

subject to

$$\begin{aligned}
 x + y/2 &\geq 2 \\
 -x/4 + y &\leq 2 \\
 x + y/2 &\leq 8 \\
 x - 2y &\leq 4 \\
 0.5u_1 - u_2 - 0.5u_3 + 2u_4 + u_5 &= -1 \\
 u_1(x + y/2 - 2) &= 0 \\
 u_2(2 + x/4 - y) &= 0 \\
 u_3(8 - x - y/2) &= 0 \\
 u_4(4 - x + 2y) &= 0 \\
 u_5(y) &= 0 \\
 x, y, u &\geq 0
 \end{aligned}$$

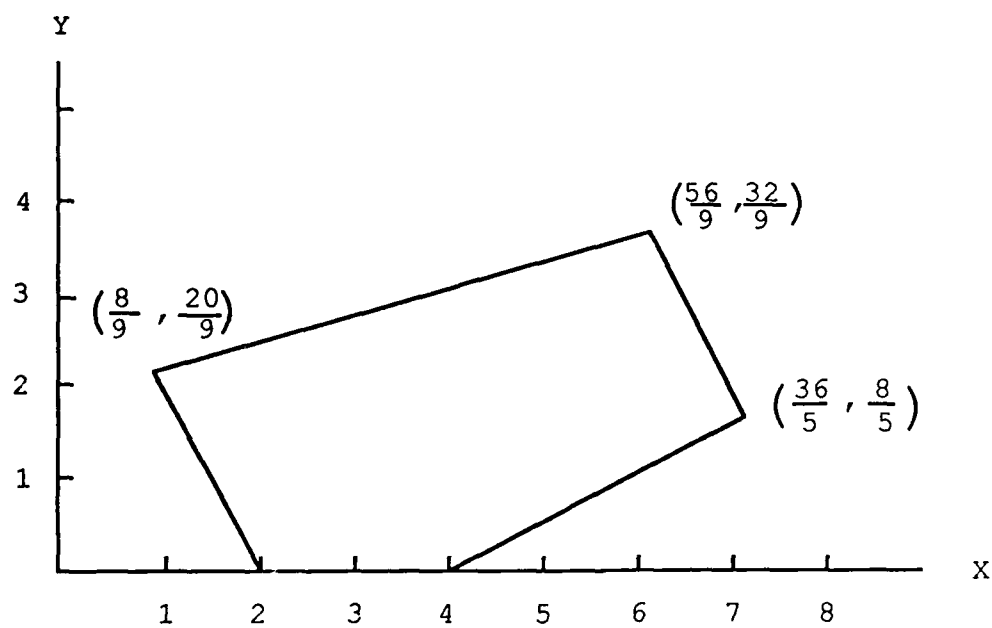


Figure 5.2: BLPP Constraint Region of Example 5.1

where u_i , $i=1, \dots, 5$, are the Kuhn-Tucker multipliers associated with the constraints containing variables controlled by the follower, and, for conciseness, let g_i , $i=1, \dots, 5$, be the corresponding functions.

The algorithm solves the above problem by explicitly setting either u_i or g_i , $i=1, \dots, 5$, to zero in a branch and bound process. At Step 1, the algorithm solves the linear program which results when the complementary slackness conditions are removed. The solution passes the test at Step 2. At Step 3, the pair (u_2, g_2) has the largest product so $S_1^+ = \{2\}$, $S_1^- = \phi$, and $S_1^0 = \{1, 3, 4, 5\}$. Going to Step 1, the new problem is solved. At Step 3 the branching variable is u_3 , so $S_2^+ = \{2, 3\}$, $S_2^- = \phi$, and $S_2^0 = \{1, 4, 5\}$. At Step 1, the new problem is infeasible, so the algorithm proceeds to Step 5. At Step 5, backtracking is accomplished and $S_2^+ = \{2\}$, $S_2^- = \{3\}$, and $S_2^0 = \{1, 4, 5\}$. Returning to Step 1, a solution is obtained, the test at Step 2 is passed, and at Step 3, the complementary slackness conditions are satisfied, so a solution in the inducible region of the BLPP has been found. The algorithm goes to Step 4 where the incumbent is updated ($\underline{F} = -8.8$). At Step 5, the

algorithm again backtracks, and $S_3^+ = \phi$, $S_3^- = \{2\}$, and $S_3^0 = \{1,3,4,5\}$. Continuing at Step 1, a solution is found, and the test at Step 2 is passed. At Step 3, the branching variable is u_3 so $S_4^+ = \{3\}$, $S_4^- = \{2\}$, and $S_4^0 = \{1,4,5\}$. At Step 1, a new solution is found which passes the test at Step 2 and satisfies the complementary slackness conditions at Step 3. At Step 4, the incumbent is updated, so $\underline{F} = -28/9$, and the algorithm backtracks at Step 5 where $S_4^+ = \phi$, $S_4^- = \{2,3\}$, and $S_4^0 = \{1,4,5\}$. At Step 1, a new solution is found, but the solution does not pass the test at Step 2. The algorithm goes to Step 5 and does not find a live vertex, so the algorithm proceeds to Step 6 and terminates with the solution $(x^*, y^*) = (8/9, 20/9)$ and $F = -28/9$. The branch and bound tree created by the algorithm has six vertices and is shown in Figure 5.3. Note each vertex in the branch and bound tree corresponds to the solution of a linear subproblem.

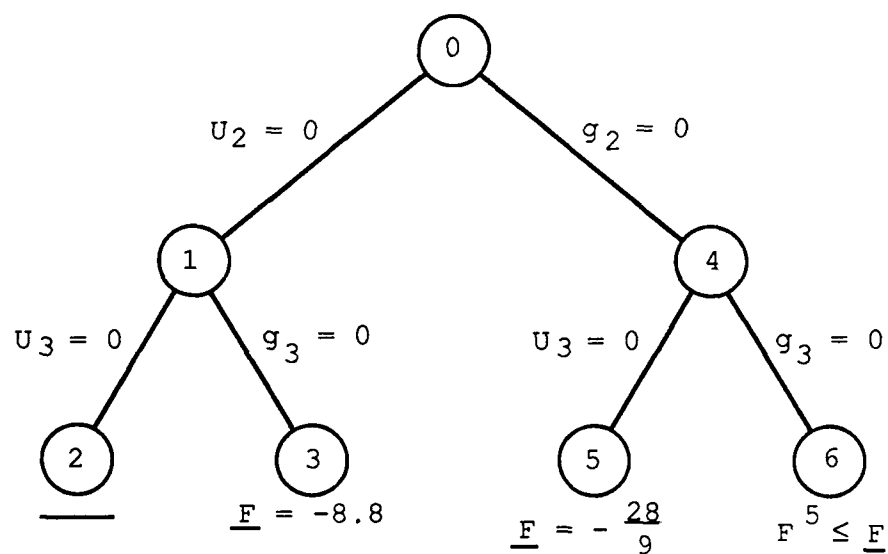


Figure 5.3: Branch and Bound Tree for Example 5.1

The example presented by Candler and Townsley (1982) to test their iterative scheme was also used by Bard and Falk (1982) to demonstrate the workings of their approach. Candler and Townsley's example has two variables controlled by the leader, three variables controlled by the follower, and three constraints. The example follows.

Example 5.2

Max $F(x,y) = 8x_1 + 4x_2 - 4y_1 + 40y_2 + 4y_3$ where y solves
 x

$$\text{Max}_y f(x,y) = -x_1 - 2x_2 - y_1 - y_2 - 2y_3$$

subject to

$$y_1 - y_2 - y_3 \geq -1$$

$$-2x_1 + y_1 - 2y_2 + 0.5y_3 \geq -1$$

$$-2x_2 - 2y_1 + y_2 + 0.5y_3 \geq -1$$

$$x \geq 0$$

$$y \geq 0$$

When this example is rewritten in its equivalent form (5.5), six Kuhn-Tucker multipliers are added to the problem. The equivalent Kuhn-Tucker representation is

$$\begin{array}{l} \text{Max } F(x,y) = 8x_1 + 4x_2 - 4y_1 + 40y_2 + 4y_3 \\ x,y,u \end{array}$$

subject to

$$y_1 - y_2 - y_3 \geq -1$$

$$-2x_1 + y_1 - 2y_2 + 0.5y_3 \geq -1$$

$$-2x_2 - 2y_1 + y_2 + 0.5y_3 \geq -1$$

$$u_1 + u_2 - 2u_3 + u_4 = 1$$

$$-1u_1 - 2u_2 + u_3 + u_5 = 1$$

$$-u_1 + 0.5u_2 + 0.5u_3 + u_6 = 2$$

$$u_1(y_1 - y_2 - y_3 + 1) = 0$$

$$u_2(-2x_1 + y_1 - 2y_2 + 0.5y_3 + 1) = 0$$

$$u_3(-2x_2 - 2y_1 + y_2 + 0.5y_3 + 1) = 0$$

$$u_4(y_1) = 0$$

$$u_5(y_2) = 0$$

$$u_6(y_3) = 0$$

$$x \geq 0, y \geq 0, u \geq 0$$

The algorithm could solve as many as 64 subproblems before terminating. In fact, the optimal solution was uncovered on the fourth iteration but was not confirmed until 10 subproblems were examined. After initializing all required data, the algorithm finds a feasible solution to the above Kuhn-Tucker representation with the complementary slackness conditions omitted, and proceeds to Step 3. The current point, $x^1 = (0,0)$, $y^1 = (1.5, 1.5, 1)$, $u^1 = (0,0,0,1,1,2)$ does not satisfy complementarity so a branching variable is selected (u_6) and the index sets are updated, giving $S_1^+ = \{6\}$, $S_1^- = \phi$, and $S_1^0 = \{1,2,3,4,5\}$. Returning to Step 1, a feasible solution is found, but the complementary slackness conditions are not satisfied at Step 3, so u_5 is picked as the branching variable with $S_2^+ = \{5,6\}$, $S_2^- = \phi$, and $S_2^0 = \{1,2,3,4\}$. Continuing at Step 1, a feasible solution is found which passes the test at Step 2. At Step 3, u_4 is chosen as the branching variable, so $S_3^+ = \{4,5,6\}$, $S_3^- = \phi$, and $S_3^0 = \{1,2,3\}$. Proceeding to Step 1, the subproblem is infeasible, so the algorithm goes to Step 5 and backtracks. At Step 5, $S_3^+ = \{5,6\}$, $S_3^- = \{4\}$, and $S_3^0 = \{1,2,3\}$. Going to Step 1, a feasible

solution is found which passes the test at Step 2 and satisfies the complementary slackness conditions at Step 3. Continuing at Step 4, $\underline{F} = 29.2$. Backtracking at Step 5, $S_4^+ = \{6\}$, $S_4^- = \{4,5\}$, and $S_4^0 = \{1,2,3\}$. Continuing at Step 1, a feasible solution is found, but at Step 2, the value of the leader's objective function for the current point is less than the incumbent lower bound, so the algorithm goes to Step 5 and backtracks, giving $S_5^+ = \emptyset$, $S_5^- = \{6\}$, and $S_5^0 = \{1,2,3,4,5\}$. Going to Step 1, the algorithm obtains a feasible solution which passes the test at Step 2. At Step 3, u_4 is selected as the branching variable, and $S_6^+ = \{4\}$, $S_6^- = \{6\}$, and $S_6^0 = \{1,2,3,5\}$. Proceeding to Step 1, a feasible solution to the current subproblem is found, Step 2 does not fathom the solution, and at Step 3, u_5 is the branching variable. $S_7^+ = \{4,5\}$, $S_7^- = \{6\}$, and $S_7^0 = \{1,2,3\}$, and going to Step 1, the new subproblem is infeasible. The algorithm goes to Step 5 and backtracks with $S_7^+ = \{4\}$, $S_7^- = \{5,6\}$, and $S_7^0 = \{1,2,3\}$. Returning to Step 1, a new solution is found which fails the test at Step 2, so backtracking is done at Step 5. $S_7^+ = \emptyset$, $S_7^- = \{4,6\}$, and $S_7^0 = \{1,2,3,5\}$, and at Step 1, the solution found fails

to be better than the current incumbent lower bound, so the algorithm proceeds to Step 5. No live vertex exists, so the the algorithm goes to Step 6 and terminates. The optimal solution is found in the fourth subproblem, and six additional subproblems are solved to guarantee no better solution exists. The optimal solution is $x = (0, 0.9)$, $y = (0, 0.6, 0.4)$, $u^* = (0, 1, 3, 6, 0, 0)$ with $F = 29.2$. The branch and bound tree for this example is shown in Figure 5.4.

By way of comparison, when this problem was solved with the separable programming approach of Bard and Falk (1982), the optimal solution was uncovered on the 51st iteration but not recognized until iteration 103. (Each iteration required the solution of a linear program in $n^1 + n^2 + 2m$ variables and $2m$ constraints.) This result typifies the relative performance of these two algorithms.

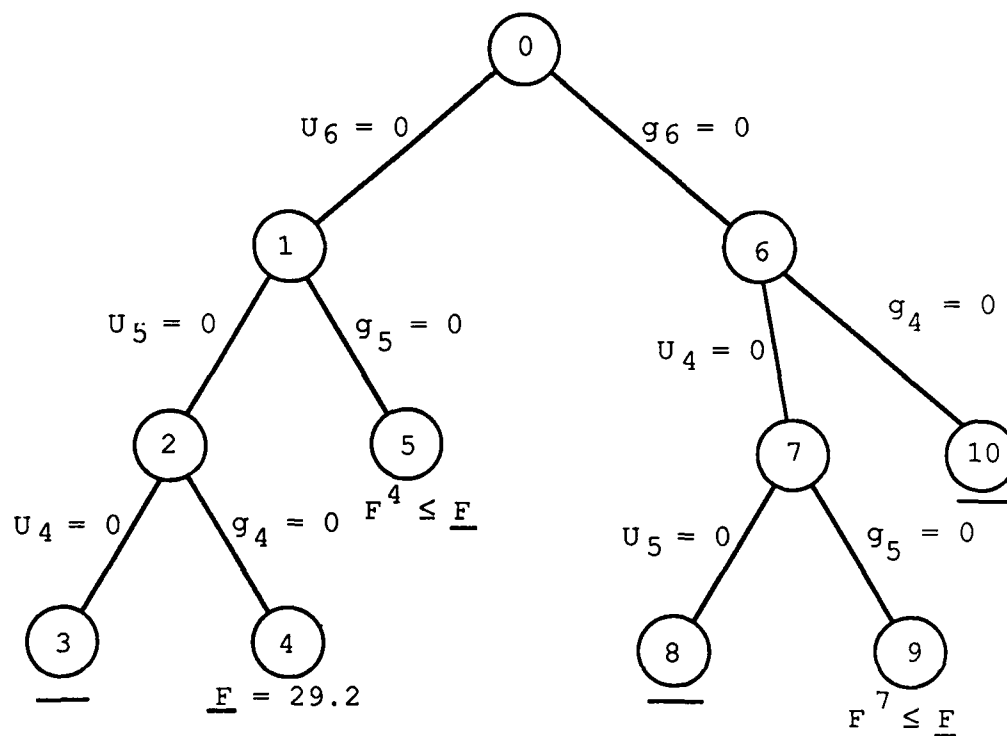


Figure 5.4: Branch and Bound Tree for Example 5.2

The third example demonstrates the algorithm's ability to solve problems with quadratic terms in the follower's objective function, an upper bounded variable, and an equality constraint. The example is shown below.

Example 5.3

Max $F(x,y) = x_1 + 2x_2 - 4y_1 + 5y_2 + 10y_3$ where y solves
 x

$$\begin{aligned} \text{Max}_{y} f(x,y) = & y_1 - y_2 - 7y_3 + 87x_1^2 + 6x_1y_1 + x_1y_3 \\ & + 8x_2y_2 + 7x_2y_3 - y_1y_3 + 5y_2^2 + 9y_2y_3 \end{aligned}$$

$$\text{subject to } x_1 + x_2 \leq 50$$

$$2x_1 + 4x_2 - y_1 = 20$$

$$3x_1 + 4y_1 + y_3 \geq 10$$

$$x_2 - 4y_2 + 2y_3 \leq 25$$

$$0 \leq y_2 \leq 100$$

$$x_1, x_2, y_1, y_3 \geq 0$$

Prior to writing the problem in the form of (5.8), the L matrix as it would appear in the follower's objective function is presented.

$$L = \begin{bmatrix} 87 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 5 & 0 \\ 1 & 7 & -1 & 9 & 0 \end{bmatrix}$$

To demonstrate the use of L , L_1 , L_2 , L_3 , L_3^s , and ℓ are shown.

$$L_1 = \begin{bmatrix} 87 & 0 \\ 0 & 0 \end{bmatrix}$$

$$L_2 = \begin{bmatrix} 6 & 0 \\ 0 & 8 \\ 1 & 7 \end{bmatrix}$$

$$L_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 5 & 0 \\ -1 & 9 & 0 \end{bmatrix}$$

$$L_3^s = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 9 \\ -1 & 9 & 0 \end{bmatrix}$$

and $\ell = (0, 5, 0)$. The equivalent representation of Example 5.3 using the Kuhn-Tucker approach follows.

$$\begin{array}{l} \text{Max } F(x,y) = x_1 + 2x_2 - 4y_1 + 5y_2 + 10y_3 \\ x,y,u \end{array}$$

subject to

$$x_1 + x_2 \leq 50$$

$$2x_1 + 4x_2 - y_1 = 20$$

$$3x_1 + 4y_1 + y_3 \geq 10$$

$$x_2 - 4y_2 + 2y_3 \leq 25$$

$$u_1 + 4u_2 + u_4 + 6x_1 - y_3 = -1$$

$$4u_3 + u_5 - u_7 + 8x_2 + 10y_2 + 9y_3 = 1$$

$$u_2 - 2u_3 + u_6 + x_1 + 7x_2 - y_1 + 9y_2 = 7$$

$$u_2(3x_1 + 4y_1 + y_3 - 10) = 0$$

$$u_3(25 - x_2 + 4y_2 - 2y_3) = 0$$

$$u_4y_1 + u_5y_2 + u_6y_3 + u_7(100 - y_2) = 0$$

$$y_2 \leq 100$$

$$x \geq 0, y \geq 0, u_1 \text{ unrestricted}, u_i \geq 0, i=2, \dots, 7$$

where u_i , $i=1, \dots, 7$, are the Kuhn-Tucker multipliers.

The first point uncovered by the algorithm at Step 1 is $x^1 = (10, 0)$ and $y^1 = (0, 100, 212.5)$ which is not recognized as bilevel feasible. The fathoming test is passed at Step 2, and at Step 3, the algorithm branches

on u_2 with $S_1^+ = \{2\}$, $S_1^- = \phi$, and $S_1^0 = \{1,3,4,5,6,7\}$.

Returning to Step 1, a new solution is found which passes the test at Step 2. At Step 3, the complementary slackness conditions are satisfied, so the algorithm proceeds to Step 4 and updates the lower bound by setting $\underline{F} = 2,635$. At Step 5, the algorithm backtracks with $S_1^+ = \phi$, $S_1^- = \{2\}$, and $S_1^0 = \{1,3,4,5,6,7\}$.

Continuing at Step 1, a new solution is found whose value is less than the current value of \underline{F} , so the corresponding vertex is fathomed at Step 2, and the algorithm proceeds to Step 5 to begin backtracking. However, there are no live vertices in the tree, so the algorithm goes to Step 6 and terminates. The optimal solution is $x^* = (10,0)$, $y^* = (0,100,212.5)$, $u^* = (0,0,451.5,151.5,0,0,4717.5)$ with $F = 2,635$.

5.5 Computational Experience

In order to test the efficiency of the algorithm, a series of problems were randomly generated and solved. In all cases, the coefficients of the A and B matrices ranged between -15 and 45 with approximately 25 percent of the coefficients being less than zero. The coefficients of the two objective functions varied between -20 and 20 with approximately 50 percent being

less than zero. The number of constraints in each problem was set at 0.4 times the total number of variables, and the right-hand side values ranged between 0 and 50. The signs of the constraints had a 0.7 probability of being \leq and a 0.3 probability of being \geq . The density of both the A and the B matrices was approximately 0.4.

All computations were performed on an IBM 3081-D using the VS Fortran compiler. The linear programming subroutine library XMP (Marsten 1981) was used to solve the linear subproblems encountered at Step 1. The primal simplex was used to solve each subproblem, and variables required to be zero were set by making their upper and lower bounds zero.

In converting a problem into a form which can be manipulated by the algorithm, slack variables are added to the constraints, g_i , $i=1, \dots, m$. These slacks are included as input and are used to convert each constraint into an equality. That is, $g_i \leq 0$, $i=1, \dots, m$, is converted into an equality by the addition of the slack variables, s_i , $i=1, \dots, m$. The constraints assume the form $g_i + s_i = 0$. In the algorithm, the

constraint g_1 is set to zero by setting the associated slack variable s_1 to zero.

5.5.1 Alternatives Explored During Testing

In the actual implementation, whenever a feasible solution is found at Step 1, the accompanying basis is stored in XMP format and used as the starting point for for the next subproblem. Two other procedures for maintaining a starting basis were tested. In the first instance, the previous basis, whether feasible or not, was used to initiate the next subproblem. Results indicate this approach is slightly worse than the one finally decided upon. In test problems, the number of subproblems solved at Step 1 ranged between 22 more and 22 less than the implemented approach. The average difference in CPU time was negligible. However, the implemented approach did appear to perform slightly better on large problems. The second approach used the basis from the last solution in the inducible region of the BLPP. It produced CPU times two to three times larger than the other approaches and was discarded.

An important determinant of computational efficiency is the rule for selecting the branching variable at Step 3. The rule chosen branches on the

Kuhn-Tucker multiplier associated with the largest complementarity term, $u_i \cdot g_i$. Other rules were examined including one that worked well for nonlinear problems. Here, the selection is made by finding the surface on which F increases most rapidly; i.e., by solving:

$$\max_i [\nabla F(x^k, y^k) \cdot \nabla g_i(x^k, y^k) / \|\nabla F(x^k, y^k)\| \cdot \|\nabla g_i(x^k, y^k)\|]$$

where ∇ is the gradient operator and $\| \cdot \|$ is the Euclidean norm. Note that when the problem is linear, the order of the components is automatically established. Using the rule and branching on g , or setting g equal to zero, resulted in large increases in CPU time and the number of subproblems solved; branching on u led to some improvement but anywhere from 50 to 100 additional subproblems had to be solved. These results were observed for moderate size problems where $n^1 = 15$, $n^2 = 10$, and $m = 10$.

After testing several other rules, it became evident that branching on g is not efficient. The branch and bound trees increase in size rapidly, so many additional subproblems are solved. In contrast, by forcing Kuhn-Tucker (KT) multipliers to zero, one of two situations quickly arises: either one of the stationarity conditions (5.5c) becomes infeasible or a

point in the inducible region emerges. For this reason, the experimentation is limited to the following branching rules

- 1) KT multiplier with largest value.
- 2) KT multiplier with smallest value.
- 3) KT multiplier associated with largest g .
- 4) KT multiplier associated with smallest g .
- 5) KT multiplier associated with smallest $u \cdot g$ product.

For moderate size problems, it was found that none of these rules performed as well as the current procedure. Increases in CPU time of 10 to 30 seconds or 30 to 200 percent were observed. Also, the number of subproblems solved showed increases up to 400.

5.5.2 Results

Table 5.1 shows the size of each problem class as defined by the total number of variables, number of variables controlled by the leader, number of variables controlled by the follower, and the number of constraints. For each problem class, 10 problems were randomly generated.

TABLE 5.1: PROBLEM SIZES

Problem Class	No. of Variables ($n^1 + n^2$)	Leader Variables (n^1)	Follower Variables (n^2)	No. of Constraints
1	40	28	12	16
2	40	24	16	16
3	50	35	15	20
4	50	30	20	20
5	50	25	25	20
6	50	20	30	20
7	70	42	28	28
8	70	35	35	28
9	90	54	36	36
10	90	45	45	36
11	100	60	40	40

Table 5.2 summarizes the computational experience with the algorithm. Each entry represents the average of 10 randomly generated problems. In all, 110 problems ranging in size from 40 to 100 variables and 16 to 40 constraints were solved. Performance measures include CPU time (seconds), the number of vertices in the branch and bound tree, and the vertex at which the optimal solution is found. Since a subproblem is solved at each vertex, the number of vertices also represents the number of subproblems solved at Step 1 of the algorithm.

TABLE 5.2: COMPUTATIONAL RESULTS

Problem Class	CPU Time (sec)	Average No. of Vertices	No. of Vertices (Range)	Optimal Solution (Vertex)
1	4.44	18	6-42	11
2	8.50	40	8-202	27
3	17.24	39	10-112	26
4	16.46	35	18-124	23
5	32.39	73	16-218	46
6	179.01	447	30-1250	391
7	106.99	96	32-270	67
8	122.26	106	26-246	84
9	352.37	138	30-384	81
10	469.40	191	132-374	147
11	294.22	159	34-476	120

As expected, the CPU time grows exponentially with the size of the problem, but more importantly, depends on the way the variables are partitioned between the players. As the number of variables controlled by the follower increases, the number of variables included in the branch and bound process increases along with the expected computational burden. Compare, for example, the two cases with 90 variables and 36 constraints. If $n^2 = 36$, there are 72 Kuhn-Tucker multipliers ($4.72 \cdot 10^{21}$ possible subproblems) while if $n^2 = 45$ there are 81 Kuhn-Tucker multipliers ($2.41 \cdot 10^{24}$ possible subproblems). This difference is reflected in the CPU

time as the case with $n^2 = 45$ requires, on average, 81 additional CPU seconds.

Also, as seen in Table 5.2, large differences in computational effort often result among problems of equivalent size. In problem class 11, 34 subproblems had to be solved at one extreme and 476 at the other. The corresponding CPU times were 85 and 804 seconds, respectively.

It is interesting to compare the above findings with those reported by others. Although Fortuny-Amat and McCarl (1981) did not seriously test their approach, some experience with their approach showed that as the size of the BLPP increased, the size of the reformulated problem increased rapidly. By testing a few moderate sized problems, it quickly became apparent that the procedure is only as good as the underlying mixed integer code. In addition, constraint redundancies in the form of $g_i = 0$ and $g_i \geq 0$ seem to place an unnecessary burden on the computations. Bialas and Karwan (1984) report results for both their "Kth-Best" algorithm and their Parametric Complementary Pivot (PCP) approach. (The former did not fair too well so it will not be discussed.) Table 5.3 presents a comparison of

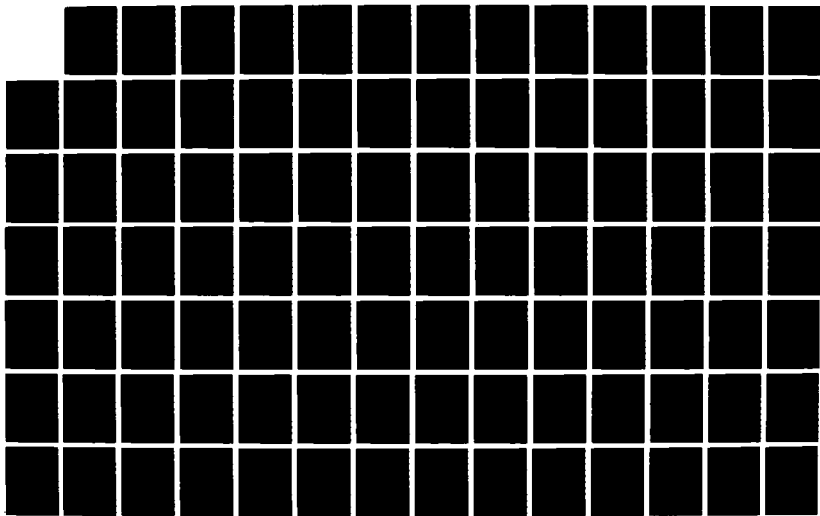
AD-A196 113

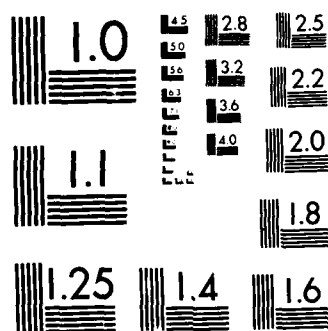
EXTENSIONS TO THE MULTILEVEL PROGRAMMING PROBLEM(U) AIR 2/4
FORCE INST OF TECH WRIGHT-PATTERSON AFB OH J T MOORE
MAY 88 AFIT/CI/MR-88-123

UNCLASSIFIED

F/G 12/5

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

the PCP algorithm and the branch and bound algorithm presented above. For each case, Bialas and Karwan solved five problems. Their computations were done on a CDC Cyber 174 using a Fortran IV code.

TABLE 5.3: COMPARISON OF COMPUTATIONAL RESULTS

No. of Variables ($n^1 + n^2$)	Follower Variables (n^2)	No. of Consts.	PCP CPU Time (sec) [†]	B&B CPU Time (sec) [†]
40	12	16	4.46	4.44
40	16	16	11.23	8.48
50	15	20	16.48	17.24
50	20	20	16.52	16.46

[†]Average of 5 problems; CDC Cyber 174

[†]Average of 10 problems; IBM 3081-D

As can be seen, the results are similar but tend to favor the branch and bound algorithm when the relative speed of the machines is taken into account. To truly determine which approach is better, a wide variety of problems should be solved using the two algorithms. It should be mentioned, however, that the PCP approach is not completely general. Its main liability centers on the requirement that the leader's objective function coefficients be nonnegative; i.e., $c^2 \geq 0$. Moreover, convergence is not guaranteed in the presence of

degeneracy, and in most instances, the algorithm only produces ε -optimal solutions.

CHAPTER 6

AN ALGORITHM FOR THE MIXED INTEGER BILEVEL PROGRAMMING PROBLEM

6.1 The Mixed Integer BLPP

The linear mixed integer bilevel programming problem (BLPP) is an interesting and challenging problem, and currently, no other algorithms exist which solve this problem. In the mixed integer BLPP, the variables are divided into four categories. Let x^1 be the n^{11} -dimensional vector of continuous variables controlled by the leader where $x^1 \in X^1 \subseteq R^{n^{11}}$, let x^2 be the n^{12} -dimensional vector of integer variables controlled by the leader where $x^2 \in X^2 \subseteq R^{n^{12}}$, let y^1 be the n^{21} -dimensional vector of continuous variables controlled by the follower where $y^1 \in Y^1 \subseteq R^{n^{21}}$, and let y^2 be the n^{22} -dimensional vector of integer variables controlled by the follower where $y^2 \in Y^2 \subseteq R^{n^{22}}$. In the above, $n^1 = n^{11} + n^{12}$, $n^2 = n^{21} + n^{22}$, and R is the set of

real numbers. Let $F(x,y)$ represent the objective function of the leader, and let $f(x,y)$ represent the objective function of the follower. Given this notation, the linear mixed integer BLPP can be written

$$\text{Max}_x F(x,y) = c^{11}x^1 + c^{12}x^2 + c^{21}y^1 + c^{22}y^2 \quad (6.1a)$$

where y solves

$$\text{Max}_y f(x,y) = d^{11}x^1 + d^{12}x^2 + d^{21}y^1 + d^{22}y^2 \quad (6.1b)$$

$$\text{subject to } A^1x^1 + A^2x^2 + B^1y^1 + B^2y^2 \leq b \quad (6.1c)$$

$$x^1, x^2, y^1, y^2 \geq 0 \quad (6.1d)$$

$$x^2, y^2 \text{ integer} \quad (6.1e)$$

where c^{11} and d^{11} are n^{11} -dimensional row vectors, c^{12} and d^{12} are n^{12} -dimensional row vectors, c^{21} and d^{21} are n^{21} -dimensional row vectors, c^{22} and d^{22} are n^{22} -

dimensional row vectors, b is an m -dimensional column vector, A^1 an $m \times n^{11}$ matrix, A^2 is an $m \times n^{12}$ matrix, B^1 is an $m \times n^{21}$ matrix, and B^2 is an $m \times n^{22}$ matrix.

Once again, because x is fixed before $f(x,y)$ is maximized, the terms $d^{11}x^1$ and $d^{12}x^2$ can be omitted from the follower's objective function without affecting the solution of (6.1). If the integer restrictions on x^2

and y^2 are relaxed, (6.1) becomes a linear BLPP, and the techniques developed in Chapter 5 can be used to find the solution of the relaxed BLPP. This trait will be utilized in creating an algorithm for the mixed integer BLPP. In the remainder of the chapter, obstacles to algorithmic development presented by the mixed integer BLPP are explained, an algorithm is presented and discussed, heuristics for solving the mixed integer BLPP are developed, an example is presented, and computational results are given.

6.2 Obstacles to Algorithmic Development

Geoffrion and Marsten (1972) have identified three key notions in the development of algorithms which solve mixed integer programming problems. They are separation, relaxation, and fathoming. These three notions are the cornerstone of the algorithm developed to solve the mixed integer BLPP. Separation, which is accomplished by placing contradictory constraints on a single integer variable, can be applied to the mixed integer BLPP. Relaxation, where all integrality requirements on the variables are dropped, can also be applied. When the integrality requirements are dropped from the mixed integer BLPP, a linear BLPP results, and

this linear BLPP can be solved using the algorithm presented in Chapter 5. However, fathoming presents several difficulties. In mixed integer programming, candidate problems or subproblems are created by separation. Relaxation is done, and some technique is employed to determine if the set of solutions of the relaxed subproblem does not include the optimal solution. If it can be ascertained that the set of solutions of the relaxed subproblem does not include a feasible solution better than the incumbent solution or the best feasible solution yet found, then the subproblem can be removed from further consideration or fathomed. Geoffrion and Marsten identify three general types of fathoming. The first type is done when the relaxed subproblem has no feasible solution. The second occurs when the solution of the relaxed subproblem is less than the value of the incumbent solution. The third type of fathoming happens when the solution of the relaxed subproblem is a feasible solution of the subproblem.

For the mixed integer BLPP, the first type of fathoming is applicable, the second type of fathoming can be done but the modifications required to

accommodate the mixed integer BLPP make it very weak, and the third type of fathoming cannot be applied. This is demonstrated in the following examples.

Example 6.1

$$\begin{array}{ll} \text{Max } F(x,y) = x + 10y & \text{where } y \text{ solves} \\ x & \end{array}$$

$$\begin{array}{l} \text{Max } f(x,y) = -y \\ y \end{array}$$

$$\text{subject to } -25x + 20y \leq 30$$

$$x + 2y \leq 10$$

$$2x - y \leq 15$$

$$2x + 10y \geq 15$$

$$x, y \geq 0$$

$$x, y \text{ integer}$$

The feasible region of this example is shown in Figure 6.1. The solution of this problem when the integrality requirements are relaxed is $(\bar{x}, \bar{y}) = (8, 1)$ with $F(\bar{x}, \bar{y}) = 18$. This solution is also in the inducible region of Example 6.1. If this example were solved using a branch and bound process, it would seem reasonable to terminate the tree at vertex 0 since the solution of the relaxed problem satisfies the integrality requirements and is in the inducible region. However, if the integrality

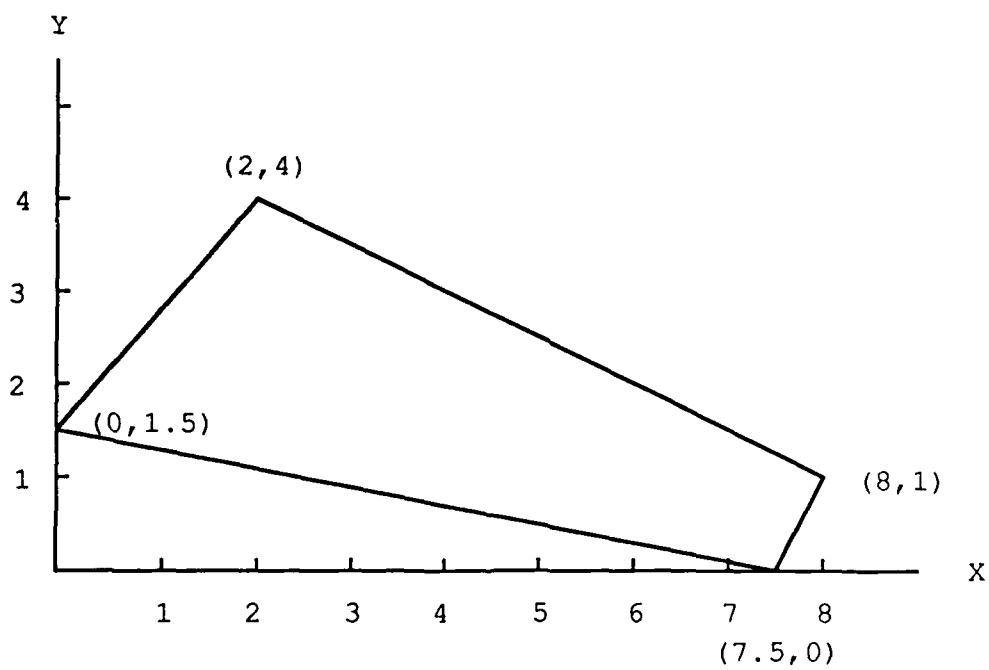


Figure 6.1: BLPP Constraint Region of Example 6.1

requirements are enforced, the optimal solution is $(x^*, y^*) = (2, 2)$ with $F(x^*, y^*) = 22$. This example shows that the solution of the relaxed BLPP may be less than the solution of the mixed integer BLPP so that the former does not provide a reliable bound on the solution of the mixed integer BLPP. The example also demonstrates that a solution which satisfies the integrality requirements and is in the inducible region of the problem cannot be fathomed in a branch and bound approach.

The following example illustrates the situation where the solution of a relaxed subproblem satisfies the integrality requirements but is not in the inducible region of the mixed integer BLPP.

Example 6.2

$$\begin{array}{ll} \text{Max}_x & F(x, y) = -x - 2y \quad \text{where } y \text{ solves} \end{array}$$

$$\begin{array}{l} \text{Max}_y & f(x, y) = y \end{array}$$

$$\text{subject to} \quad -x + 2.5y \leq 3.75$$

$$x + 2.5y \geq 3.75$$

$$2.5x + y \leq 8.75$$

$$x, y \geq 0$$

$$x, y \text{ integer}$$

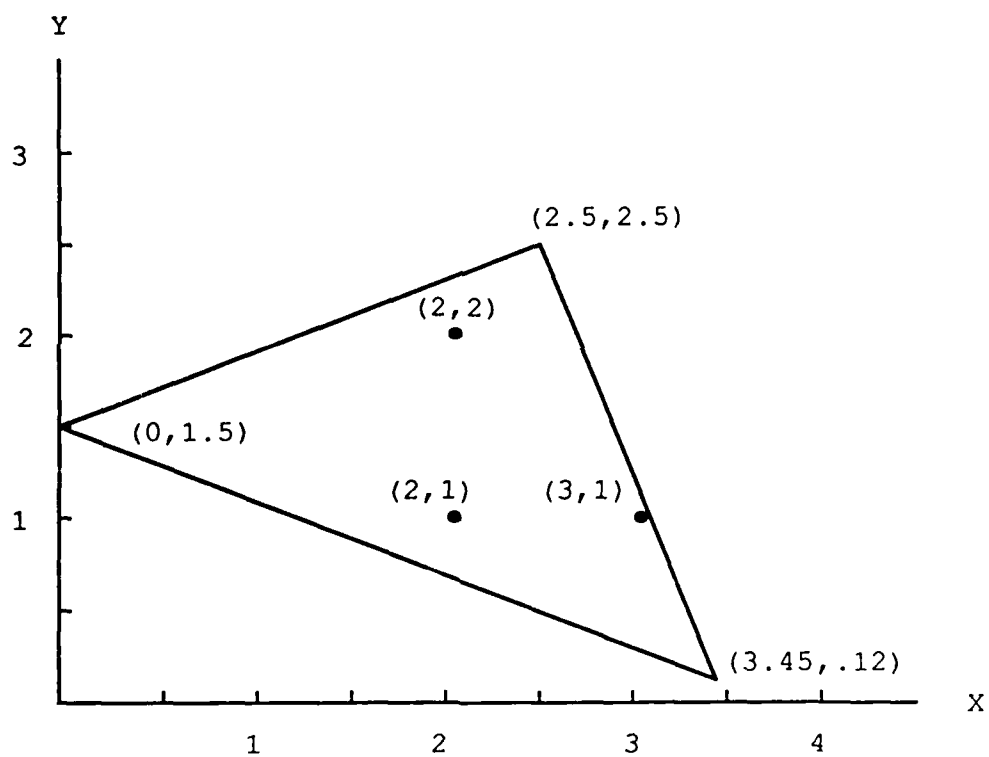
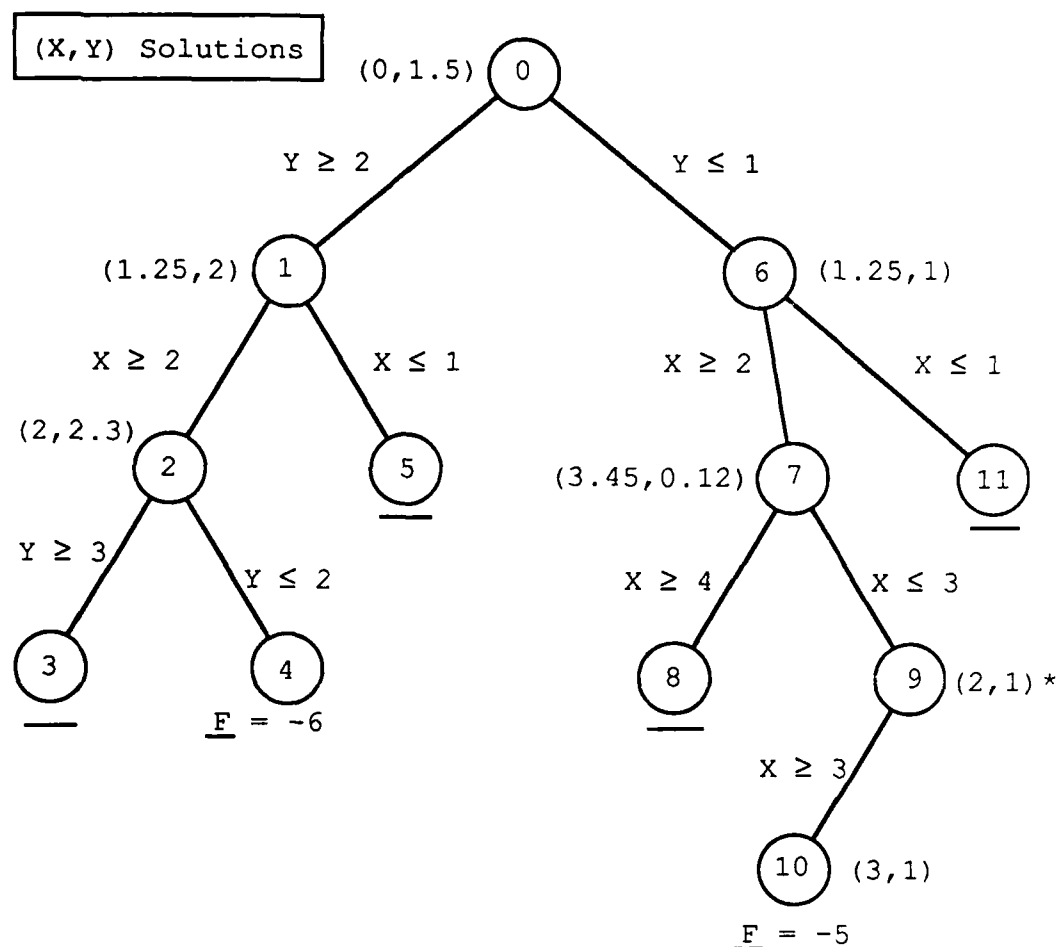


Figure 6.2: BLPP Constraint Region of Example 6.2

As shown in Figure 6.2, the feasible region of Example 6.2 contains three integer points. They are $(2,1)$, $(2,2)$, and $(3,1)$. If the leader picks $x = 2$, the follower chooses $y = 2$, so $F = -6$. If the leader's decision is $x = 3$, the follower's choice is $y = 1$, so $F = -5$. Therefore, the optimal solution of the problem is $(x^*, y^*) = (3,1)$ with $F = -5$.

Now consider a separation, relaxation, and fathoming approach to Example 6.2 using a depth first search. Separation is done by forming subproblems with contradictory constraints on a single integer variable, and relaxation is accomplished by ignoring the integrality requirements on the variables. Thus, each subproblem is a mixed integer BLPP with tighter bounds placed on the integer variables. To facilitate the presentation, a branch and bound tree is used. The contradictory constraints appear on the branches of the tree, and the vertices of the tree represent the subproblems formed by adding the constraints along the branches to the problem. The branching is done on a variable with a fractional part and is done in the \geq direction. The tree is presented in Figure 6.3.



*Integer valued but not in the inducible region of the BLPP.

Figure 6.3: Branch and Bound Tree for Example 6.2

At vertex 0, the solution of the relaxation of Example 6.2 is $(x,y) = (0,1.5)$ with $F = -3$. Separating on the y variable and adding the constraint $y \geq 2$ to Example 6.2 yields the subproblem at vertex 1. Solving the relaxation of the subproblem at vertex 1 yields the solution $(x,y) = (1.25,2)$ with $F = -5.25$. Separating on the variable x and adding the constraint $x \geq 2$ to the subproblem at vertex 1 produces a more constrained subproblem at vertex 2. The solution of the relaxation of the subproblem at vertex 2 is $(x,y) = (2,2.3)$ with $F = -6.6$. Separating on the variable y and adding the contradictory constraints $y \geq 3$ and $y \leq 2$ produce, respectively, the subproblems at vertices 3 and 4. The subproblem at vertex 3 is infeasible. Proceeding to the unexplored subproblem at vertex 4, relaxing, and solving gives $(x,y) = (2,2)$ with $F = -6$. This solution is in the inducible region of the subproblem at vertex 4 and is also in the inducible region of Example 6.2, so the solution $(x,y) = (2,2)$ becomes the incumbent solution. At vertex 4, the subproblem has constraints on y which force $y = 2$. Given $y = 2$, the leader's decision is $x = 2$. This allows fathoming of the subproblem at vertex 4, and backtracking in the tree leads to the infeasible

subproblem at vertex 5. Backtracking again returns to the problem at vertex 0. Adding the contradictory constraint $y \leq 1$ creates the subproblem at vertex 6. The solution of its relaxation yields $F = -3.25$ with $(x,y) = (1.25,1)$. Separating on the x variable creates the subproblem at vertex 7 with the constraint $x \geq 2$ added. The solution of the relaxed subproblem at vertex 7 is $(x,y) = (3.45,0.12)$ with $F = -3.69$. Separating on x and adding the constraint $x \geq 4$ produces an infeasible subproblem at vertex 8. Backtracking and adding the contradictory constraint $x \leq 3$ to the subproblem at vertex 7 gives the subproblem at vertex 9. Its solution is $(x,y) = (2,1)$ with $F = -4$. This is in the inducible region of the subproblem at vertex 9. However, $(x,y) = (2,1)$ is not in the inducible region of Example 6.2, and if the subproblem at vertex 9 is fathomed, the optimal solution is not found. The variable x must be further constrained in order to uncover the optimal solution of Example 6.2. The subproblem at vertex 10 with the constraint $x \geq 3$ added must be explored. Solving the relaxation of this subproblem gives the solution $(x,y) = (3,1)$ with $F = -5$. This is in the inducible region of the subproblem at vertex 10 and is also in the inducible

region of Example 6.2. As previously shown, this is also the optimal solution of Example 6.2. Backtracking adds vertex 11 to the branch and bound tree. The subproblem at vertex 11 is infeasible, and the branch and bound tree is complete.

Thus, Example 6.2 demonstrates that a solution in the inducible region of a subproblem may not be in the inducible region of the original problem and that a subproblem cannot be fathomed when a solution satisfies the integrality requirements but is not in the inducible region of the original problem. At vertex 7, if y had been selected as the branching variable, the optimal solution would not have been found.

These two examples show that a subproblem cannot be fathomed when the solution of its relaxation satisfies the integrality requirements or is in the inducible region of the original problem. These examples also show that the solution of the relaxed problem cannot be used as an upper bound on the solution of the mixed integer BLPP.

As stated earlier, the second type of fathoming can be applied in solving the mixed integer BLPP. The second type of fathoming may be done when the value of

the solution of a relaxed subproblem is less than the value of the incumbent solution. However, for the mixed integer BLPP where the relaxation is limited to the dropping of the integrality requirements, Examples 6.1 and 6.2 show that if this type of fathoming is done, the optimal solution of the mixed integer BLPP may not be found. However, further relaxation of a subproblem does permit this type of fathoming to be done. Before discussing this situation, some definitions and notation are presented.

6.3 Definitions and Notation

Before formally presenting the situation where the second type of fathoming is applicable, some notation and definitions must be presented. Let

$N = \{1, \dots, n^1 + n^2\}$ be the index set of the variables;

$N^1 = \{1, \dots, n^{12}\}$ be index set of the integer variables, x^2 , controlled by the leader;

$N^2 = \{1, \dots, n^{22}\}$ be the index set of integer variables, y^2 , controlled by the follower.

The initial upper bounds placed on the integer variables of (6.1), the mixed integer BLPP, are defined by the following vectors. U^1 is an n^{12} -dimensional vector of

upper bounds on the integer variables controlled by the leader, and U^2 is an n^{22} -dimensional vector of upper bounds on the integer variables controlled by the follower. If an integer variable is unbounded above, then the corresponding entry in the upper bound vector is ∞ . The initial lower bound on each integer variable of (6.1) is assumed to be zero. For subproblem k , the sets of bounds on the variables are

$$H_k^1 = \{(\alpha^{1k}, \beta^{1k}) \mid 0 \leq \alpha_j^{1k} \leq x_j^2 \leq \beta_j^{1k} \leq U_j^1, j \in N^1\}$$

$$H_k^2 = \{(\alpha^{2k}, \beta^{2k}) \mid 0 \leq \alpha_j^{2k} \leq y_j^2 \leq \beta_j^{2k} \leq U_j^2, j \in N^2\}$$

where, for subproblem k , α^{1k} and β^{1k} are n^{12} -dimensional vectors of lower and upper bounds, respectively, placed on the integer variables controlled by the leader, and α^{2k} and β^{2k} are n^{22} -dimensional vectors of lower and upper bounds, respectively, placed on the integer variables controlled by the follower. When no positive lower bound has been placed on an integer variable of subproblem k , then the entry in the appropriate lower bound vector is zero. Also, for subproblem k , the notation $H_k^2(0, \infty)$ is used to indicate that no bounds other than the original bounds imposed in the mixed

integer BLPP (6.1) are placed on the integer variables controlled by the follower. In addition, the notation $H_\ell^1 \subseteq H_k^1$ means $\alpha_j^{1k} \leq \alpha_j^{1\ell}$ and $\beta_j^{1\ell} \leq \beta_j^{1k}$.

The index sets of the integer variables which are restricted in subproblem k are

$$S_k^1 = \{j \mid \alpha_j^{1k} > 0 \text{ or } \beta_j^{1k} < U_j^1, j \in N^1\}$$

$$S_k^2 = \{j \mid \alpha_j^{2k} > 0 \text{ or } \beta_j^{2k} < U_j^2, j \in N^2\}.$$

As implied by the definition of S_k^1 and S_k^2 , a restricted variable is a variable which has an additional lower or upper bound placed on it in subproblem k . When a variable is referred to as restricted, this means constraints or bounds other than those found in (6.1) are placed on the variable.

As noted in the previous section, the second type of fathoming may be done if a subproblem is sufficiently relaxed. The relaxed BLPP is the BLPP derived from the mixed integer BLPP by dropping the integrality requirements. The following notation is used to designate the solution of the relaxed BLPP. Define F_k^C as the optimal solution of the relaxed or continuous BLPP at subproblem k . F_k^C is found by dropping the

integrality requirements of subproblem k and solving the resulting BLPP where all variables are continuous. The BLPP which is solved to find F_k^C is

$$\text{Max}_x F(x,y) = c^{11}x^1 + c^{12}x^2 + c^{21}y^1 + c^{22}y^2 \quad (6.2a)$$

where y solves

$$\text{Max}_y f(x,y) = d^{21}y^1 + d^{22}y^2 \quad (6.2b)$$

$$\text{subject to } A^1x^1 + A^2x^2 + B^1y^1 + B^2y^2 \leq b \quad (6.2c)$$

$$x^1, x^2, y^1, y^2 \geq 0 \quad (6.2d)$$

$$\text{and the bounds } H_k^1 \text{ and } H_k^2 \quad (6.2e)$$

As shown in Chapter 5, the BLPP can be restated in an equivalent form which utilizes the Kuhn-Tucker conditions of the follower's problem. If the BLPP is relaxed by dropping the Kuhn-Tucker conditions, the solution of the resulting problem is called the "high point" solution of the relaxed BLPP. Problem (6.3) illustrates the problem which is solved to find the "high point" solution of subproblem k .

$$\text{Max}_{x,y} F(x,y) = c^{11}x^1 + c^{12}x^2 + c^{21}y^1 + c^{22}y^2 \quad (6.3a)$$

$$\text{subject to } A^1x^1 + A^2x^2 + B^1y^1 + B^2y^2 \leq b \quad (6.3b)$$

$$x^1, x^2, y^1, y^2 \geq 0 \quad (6.3c)$$

$$\text{and the bounds } H_k^1 \text{ and } H_k^2 \quad (6.3d)$$

The above problem is referred to as the high point problem. Define F_k^H as the "high point" solution of the high point problem derived from subproblem k and note that problem (6.3) is derived from the relaxed BLPP (6.2).

6.4 Bounds which Allow the Second Type of Fathoming

Two theorems and one corollary are presented. They state the conditions that must be satisfied so that the solution of a relaxed subproblem may be used as a bound on the value of the mixed integer BLPP. The bound allows fathoming of a subproblem when the value of the bound is less than or equal to the value of the incumbent solution.

Before presenting the theorems and corollary, a method which allows easy and simple presentation of the subproblems is required. Branch and bound provides a

means to keep track of the subproblems and the contradictory constraints which are added to (6.1). Each vertex of the branch and bound tree represents a new subproblem, and each branch of the tree represents an added constraint on one of the integer variables. If vertex k is along the path to vertex l , this means the subproblem associated with vertex l is derived from the subproblem associated with vertex k , so $H_l^1 \subseteq H_k^1$ and $H_l^2 \subseteq H_k^2$.

Theorem 6.1 Given H_k^1 and $H_k^2(0, \infty)$, let (\bar{x}^k, \bar{y}^k) be the high point solution to the corresponding relaxed BLPP. Then $F_k^H = F(\bar{x}^k, \bar{y}^k)$ is an upper bound on the solution of the mixed integer BLPP at vertex k .

Proof: Let $(\bar{x}^{*l}, \bar{y}^{*l})$ solve the mixed integer BLPP at vertex l where $H_l^1 \subseteq H_k^1$ and $H_l^2 \subseteq H_k^2(0, \infty)$. Assume $F(\bar{x}^{*l}, \bar{y}^{*l}) > F_k^H$. However, this leads to a contradiction because $(\bar{x}^{*l}, \bar{y}^{*l})$ is a feasible solution of the high point problem at vertex k . ■

This result says that the high point solution at vertex k may be used as a bound in determining if the subproblem at vertex k can be fathomed. Unfortunately,

this bound is only applicable when no restrictions on the integer variables controlled by the follower have been made along the path to vertex k . Thus, Theorem 6.1 does not hold if the path to vertex k contains a restriction on one of the integer variables controlled by the follower; this means Theorem 6.1 does not hold if $S_k^2 \neq \emptyset$. The reason for this is that the follower does not have to adhere to any restrictions placed on his variables in the restricted subproblems. Once the leader has made a decision, the follower optimizes his objective function without regard to the restrictions, S_k^2 , placed on the variables he controls. Nevertheless, the following theorem indicates when F_k^H provides a valid upper bound for the case where restrictions have been placed on the integer variables controlled by the follower.

Theorem 6.2 Given H_k^1 and H_k^2 , let $(\tilde{x}^k, \tilde{y}^k)$ be the high point solution to the corresponding relaxed BLPP. Then $F_k^H = F(\tilde{x}^k, \tilde{y}^k)$ is an upper bound on the mixed integer BLPP defined by the current path in the tree if none of the y_j^{2k} are at either $\alpha_j^{2k} > 0$ or $\beta_j^{2k} < U_j^2$ for $j \in S_k^2$.

Proof: Let $(\underline{x}^{*l}, \underline{y}^{*l})$ solve the mixed integer BLPP at vertex l where $H_l^1 \subseteq H_k^1$ and $H_l^2 \subseteq H_k^2$. Assume $F(\underline{x}^{*l}, \underline{y}^{*l}) > F_k^H$. Thus $(\underline{x}^{*l}, \underline{y}^{*l})$ cannot be feasible to the high point problem at vertex k . Now construct the line joining $(\underline{x}^k, \underline{y}^k)$ and $(\underline{x}^{*l}, \underline{y}^{*l})$ as follows: let $(\bar{x}, \bar{y}) = \gamma(\underline{x}^k, \underline{y}^k) + (1-\gamma)(\underline{x}^{*l}, \underline{y}^{*l})$, $\gamma \in [0, 1]$. So for $\gamma > 0$, $F(\bar{x}, \bar{y}) = F[\gamma(\underline{x}^k, \underline{y}^k) + (1-\gamma)(\underline{x}^{*l}, \underline{y}^{*l})] = \gamma F(\underline{x}^k, \underline{y}^k) + (1-\gamma)F(\underline{x}^{*l}, \underline{y}^{*l})$ since F is linear. It is assumed $F(\underline{x}^{*l}, \underline{y}^{*l}) > F_k^H$, so $\gamma F(\underline{x}^k, \underline{y}^k) + (1-\gamma)F(\underline{x}^{*l}, \underline{y}^{*l}) > \gamma F(\underline{x}^k, \underline{y}^k) + (1-\gamma)F(\underline{x}^k, \underline{y}^k) = F(\underline{x}^k, \underline{y}^k)$. Thus, $F(\bar{x}, \bar{y}) > F(\underline{x}^k, \underline{y}^k)$. However, this contradicts the optimality of $(\underline{x}^k, \underline{y}^k)$ because for γ sufficiently small, (\bar{x}, \bar{y}) is feasible to the high point problem. ■

The weakness of Theorem 6.2 arises from the fact that the conditions required for the theorem to be true will not usually hold. In solving for the high point solution, the requirement that none of the restricted integer variables controlled by the follower be at a bound will usually be violated. However, the following corollary provides a way to always find a bound which may be used to do fathoming.

Corollary 6.1 Given H_k^1 and H_k^2 , let $(\tilde{x}^k, \tilde{y}^k)$ be the high point solution of the corresponding relaxed BLPP with the restrictions of H_k^2 relaxed. Then $F_k^H = F(\tilde{x}^k, \tilde{y}^k)$ is an upper bound on the mixed integer BLPP defined by the current path in the tree.

Proof: Relaxing the restrictions of H_k^2 is equivalent to replacing H_k^2 with $H_k^2(0, \infty)$. Thus, Theorem 6.1 applies. ■

In Corollary 6.1, problem (6.3) is modified before the high point solution is found. This modification requires a change in the constraints of (6.3d). The modification is the relaxing of the restrictions of H_k^2 . Thus, the problem of interest in Corollary 6.1 is

$$\text{Max}_{x,y} F(x,y) = c^{11}x^1 + c^{12}x^2 + c^{21}y^1 + c^{22}y^2 \quad (6.4a)$$

$$\text{subject to } A^1x^1 + A^2x^2 + B^1y^1 + B^2y^2 \leq b \quad (6.4b)$$

$$x^1, x^2, y^1, y^2 \geq 0 \quad (6.4c)$$

$$\text{and the bounds } H_k^1 \text{ and } H_k^2(0, \infty) \quad (6.4d)$$

The bounds presented above are weak. In the BLPP, once the leader has made his decision, the follower is

free to respond without regard to any restrictions used to reach the leader's decision. This precludes the use of tighter bounds in finding the solution of the mixed integer BLPP.

6.5 A Modification of the Algorithm which Solves the BLPP

In Chapter 5, an algorithm is developed which solves the BLPP. This algorithm applies a branch and bound approach to an equivalent representation of the BLPP. This representation is derived by using the Kuhn-Tucker conditions of the follower's problem. The stationarity conditions are then added, and a branch and bound procedure is applied to find solutions that satisfy the complementary slackness conditions. This algorithm can be used to solve the relaxed mixed integer BLPP. However, when there are restrictions on the integer variables controlled by the follower, the algorithm that solves the BLPP must be modified before it can be used to solve the relaxed mixed integer BLPP. The modification is given by the following proposition.

Proposition 6.1: When solving the relaxed mixed integer BLPP, the complementary slackness condition associated with an integer variable controlled by the follower should be ignored when that variable is at an upper bound.

Proposition 6.1 means that if, in solving the relaxed mixed integer BLPP at vertex k , $y_j^{2k} = \beta_j^{2k} \leq U_j^2$ for $j \in N^2$, then in the algorithm which solves the BLPP, the complementary slackness condition associated with that variable should be relaxed (ignored). Example 6.3 illustrates when this situation arises.

Example 6.3

$$\begin{array}{ll}
 \text{Max}_{x,y} F(x,y) = -2x + 15y & \text{where } y \text{ solves} \\
 \text{Max}_{y} f(y) = -10y & \\
 \text{subject to} & x \geq 19.25 \\
 & y \leq 14 \\
 & 11.3x - 7.8y \leq 116 \\
 & x, y \geq 0 \\
 & y \text{ integer.}
 \end{array}$$

A graph of the feasible region of Example 6.3 appears in Figure 6.4. When the integrality requirement on y in Example 6.3 is dropped, the relaxed mixed integer BLPP

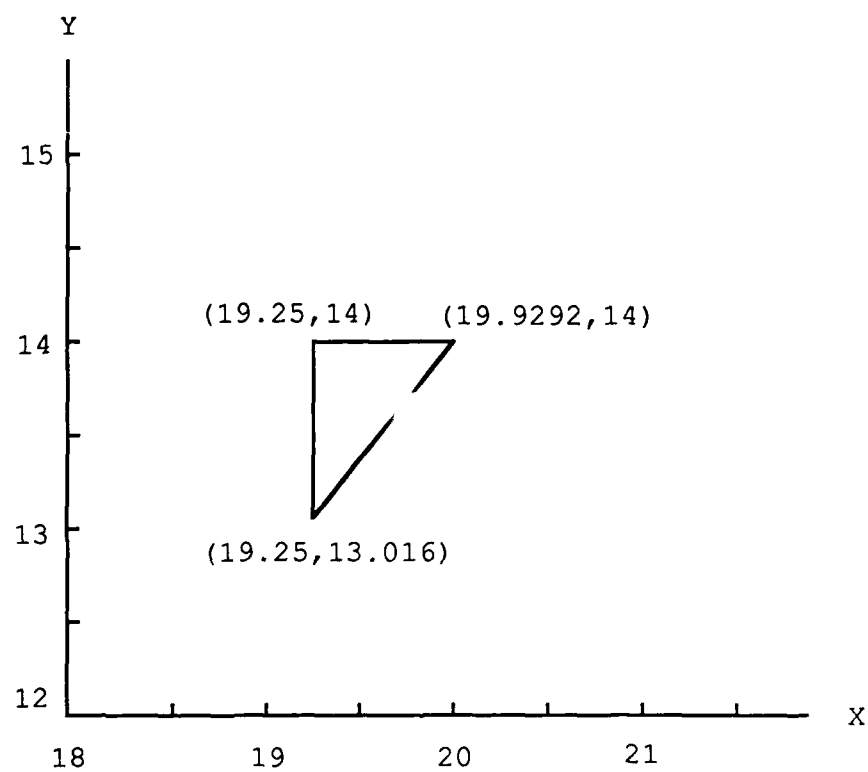


Figure 6.4: BLPP Constraint Region of Example 6.3

can be rewritten using the Kuhn-Tucker approach. Its equivalent representation is

$$\begin{array}{ll} \text{Max } F(x,y) = -2x + 15y & (6.5a) \\ x,y,u \end{array}$$

$$\text{subject to } x \geq 19.25 \quad (6.5b)$$

$$y \leq 14 \quad (6.5c)$$

$$11.3x - 7.8y \leq 116 \quad (6.5d)$$

$$-u_1 + 7.8u_2 + u_3 = 10 \quad (6.5e)$$

$$u_1(14 - y) = 0 \quad (6.5f)$$

$$u_2(116 - 11.3x + 7.8y) = 0 \quad (6.5g)$$

$$u_3(y) = 0 \quad (6.5h)$$

$$x, y, u_1, u_2, u_3 \geq 0 \quad (6.5i)$$

where u_1 is the Kuhn-Tucker (KT) multiplier associated with (6.5c), u_2 is the KT multiplier associated with (6.5d), and u_3 is the KT multiplier associated with the nonnegativity requirement of y . When the algorithm of Chapter 5 is applied to the above problem, the solution is $(x^*, y^*) = (19.9292, 14)$ with $F = 170.14$. However, if the leader chooses $x = 19.25$, and the integrality requirement on the follower's response is enforced, the follower's decision is $y = 14$. For $(x, y) = (19.25, 14)$,

$F = 171.5$. The algorithm does not reach the point $(19.25, 14)$ because the stationarity condition (6.5e) requires u_2 or u_3 to be positive. If u_3 is positive, y must be zero to satisfy the complementary slackness condition (6.5h). This is not feasible, so u_2 takes on positive value. The complementary slackness condition (6.5g) associated with u_2 requires the constraint (6.5d) to be tight, and x must be 19.9292 in order to satisfy this requirement. However, if the complementary slackness condition (6.5h) associated with y is relaxed, the solution is $(x, y) = (19.25, 14)$. This happens because both u_3 and y can have values greater than zero. Since u_3 can be greater than zero, this allows u_2 to assume a value of zero which means constraint (6.5d) does not have to be tight. Thus, when an upper bounded integer variable controlled by the follower is at its upper bound, the complementary slackness condition associated with this variable in the equivalent representation of the relaxed mixed integer BLPP must be relaxed.

This requires a slight change in Step 3 of the algorithm which solves the BLPP. Step 3 is

Step 3: (Branching) If $u_i^k \cdot g_i(x^k, y^k) = 0$, $i=1, \dots, m+n^2$, go to Step 4; otherwise, select i for which $u_i^k \cdot g_i(x^k, y^k)$ is largest and label it i_1 . Put $S_k^+ \leftarrow S_k^+ \cup \{i_1\}$, $S_k^0 \leftarrow S_k^0 \setminus \{i_1\}$, $S_k^- \leftarrow S_k^-$, and go to Step 1.

The modified step is

Step 3: (Branching) If y_j^k , $j=1, \dots, n^2$, is an integer variable at its upper bound, then delete the i from the list $i=1, \dots, m+n^2$ which corresponds to this variable's complementary slackness condition. If $u_i^k \cdot g_i(x^k, y^k) = 0$, $i=1, \dots, m+n^2$, go to Step 4; otherwise, select i for which $u_i^k \cdot g_i(x^k, y^k)$ is largest and label it i_1 . Put $S_k^+ \leftarrow S_k^+ \cup \{i_1\}$, $S_k^0 \leftarrow S_k^0 \setminus \{i_1\}$, $S_k^- \leftarrow S_k^-$, and go to Step 1.

This modification allows the leader to take advantage of the integrality requirement on variables controlled by the follower. With this modification to the algorithm for solving the BLPP, an algorithm which finds feasible solutions of the mixed integer BLPP can be presented.

6.6 An Algorithm for the Mixed Integer BLPP

The algorithm developed for solving the mixed integer BLPP takes a branch and bound approach which utilizes a standard branching procedure. On each pass through the algorithm, an integer variable which is fractional-valued in the solution of the relaxed mixed integer BLPP is selected, and separation is based on this variable. When separation is accomplished, two subproblems are created. The subproblems are generated by adding alternative constraints on the selected integer variable. Assuming x_j^2 is the selected integer variable, the alternative constraints are $x_j^2 \leq \lfloor x_j^2 \rfloor$ and $x_j^2 \geq \lfloor x_j^2 \rfloor + 1$ where $\lfloor \cdot \rfloor$ represents the greatest integer value less than or equal to that of the selected variable. Adding these constraints is equivalent to placing bounds on the integer variables, so in the algorithm, the alternative constraints are added by placing bounds on the integer variables. By treating the added constraints as bounds, the algorithm can use H_k^1 , H_k^2 , S_k^1 , and S_k^2 to do the bookkeeping required to maintain the branch and bound tree. Fathoming is done when the relaxed BLPP is infeasible or when the upper bound on the mixed integer BLPP at a vertex is less than

the value of the current incumbent solution. By applying Corollary 6.1, the high point solution is used as an upper bound on the solution of the subproblems. That is, the restrictions or bounds of H_k^2 placed on the integer variables controlled by the follower are ignored in finding the high point solution at vertex k . In the algorithm, \underline{F} is the value of the incumbent solution. A flow chart of the algorithm is shown in Figure 6.5. As will be shown, the algorithm cannot find the optimal solution of all mixed integer BLPPs. However, the algorithm always finds a solution which is in the inducible region of the mixed integer BLPP. Also, \underline{x} will represent the vector (x^1, x^2) , and \underline{y} will represent the vector (y^1, y^2) .

6.6.1 The Basic Algorithm

Step 0: (Initialization) Put $k=0$. Set the bounds of H_k^1 and H_k^2 to the bounds of the mixed integer BLPP. Put $S_k^1 = \phi$, $S_k^2 = \phi$, $\underline{F} = -\infty$.

- Step 1: (High Point Solution) Attempt to find the high point solution of (6.4) and calculate F_k^H . If infeasible or $F_k^H \leq \underline{F}$, go to Step 6.
- Step 2: (Relaxed Solution) Attempt to solve the relaxed BLPP (6.2). If infeasible, go to Step 6. If successful, label the solution $(\underline{x}^k, \underline{y}^k)$.
- Step 3: (Branching) If the integer requirements of (6.1) are satisfied by $(\underline{x}^k, \underline{y}^k)$, go to Step 4. Otherwise, select an x_j^{2k} , $j \in N^1$, or y_j^{2k} , $j \in N^2$, which is fractional-valued. Place a new bound on the selected variable. Put $k \leftarrow k+1$, add a live vertex to the branch and bound tree, and update H_k^1 , H_k^2 , S_k^1 , and S_k^2 . Go to Step 1.
- Step 4: (Feasible Point) Fix x at \underline{x}^k and solve the follower's problem to obtain $(\underline{x}^k, \hat{\underline{y}}^k)$. Compute $F(\underline{x}^k, \hat{\underline{y}}^k)$ and put $\underline{F} = \max[\underline{F}, F(\underline{x}^k, \hat{\underline{y}}^k)]$.

Step 5: (Integer Branching) If $\alpha_j^{1k} = \beta_j^{1k}$, for each $j \in N^1$, and $\alpha_j^{2k} = \beta_j^{2k}$, for each $j \in N^2$, go to Step 6. Otherwise, select an integer variable such that $\alpha_j^{1k} \neq \beta_j^{1k}$, $j \in N^1$, or $\alpha_j^{2k} \neq \beta_j^{2k}$, $j \in N^2$, and place a new bound on it. Put $k \leftarrow k+1$, add a live vertex to the branch and bound tree, and update H_k^1 , H_k^2 , S_k^1 , and S_k^2 . Go to Step 1.

Step 6: (Backtracking) If no live vertex exists, go to Step 7. Otherwise, branch to the newest live vertex, put $k \leftarrow k+1$, and update H_k^1 , H_k^2 , S_k^1 , and S_k^2 . Go to Step 1.

Step 7: (Termination) If $\underline{F} = -\infty$, there is no feasible solution to (6.1). Otherwise, terminate with the feasible solution associated with \underline{F} .

At Step 0, H_k^1 and H_k^2 are initialized by placing the lower bounds on the integer variables in the vectors α_k^1 and α_k^2 and the upper bounds on the integer variables in the vectors β_k^1 and β_k^2 . At Step 1, problem (6.4) is solved to find the high point solution at vertex k . If (6.4) is infeasible, the algorithm goes to Step 6. At

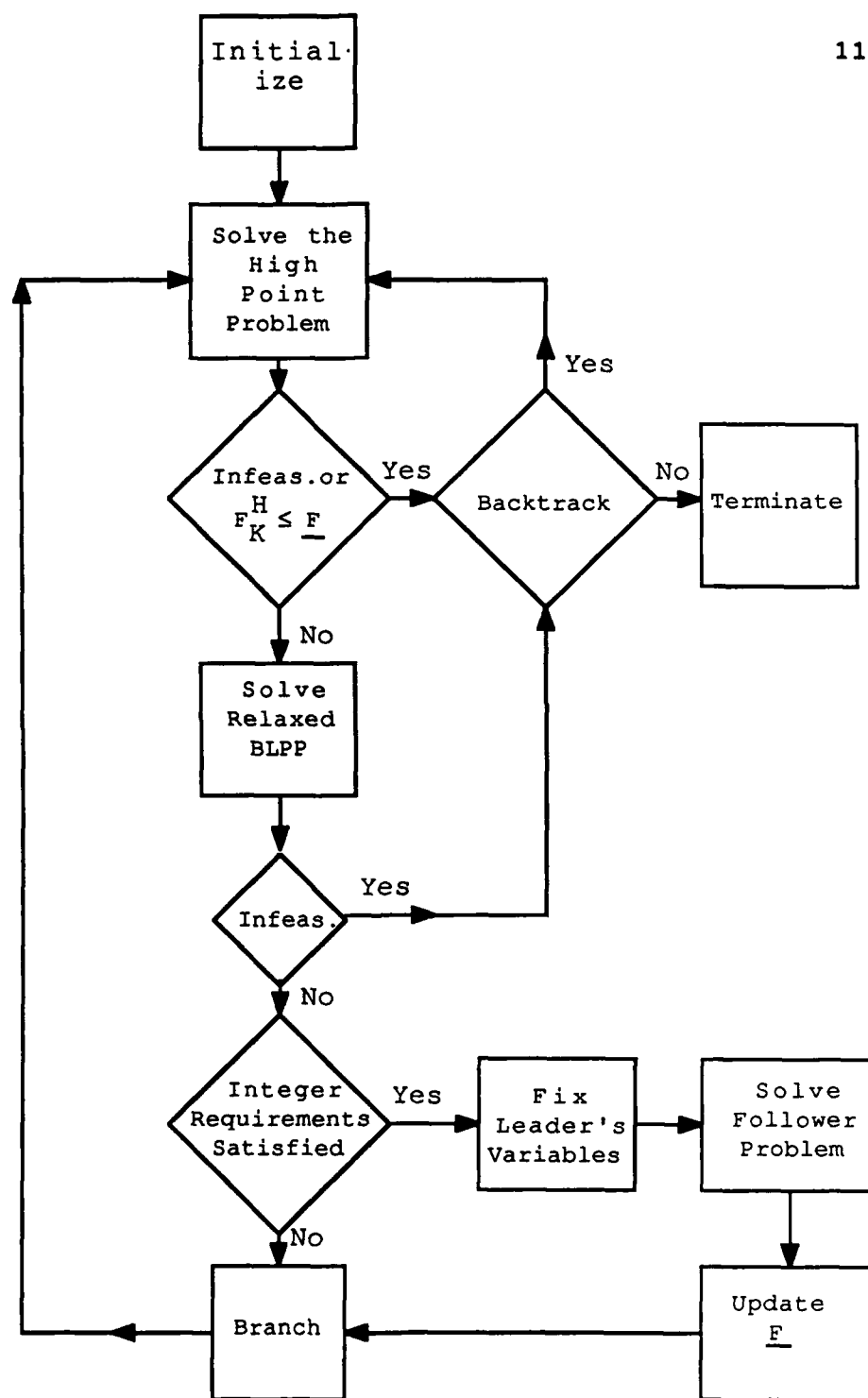


Figure 6.5: Flow Chart of Mixed Integer BLPP Algorithm

Step 2, the relaxed BLPP at vertex k is solved. If the relaxed BLPP does not have a feasible solution, the algorithm skips to Step 6.

At Step 3, many rules for selecting the branching variable were considered. The rule which was selected for implementation is the one which had the shortest average CPU time for 10 test problems. So, at Step 3, if at least one integer variable has a fractional value in the solution found at Step 2, branching is accomplished. The variable selected for branching is the integer variable with the largest fractional part. That is, the variable selected for branching is an integer variable such that the value of $(x_j^{2k} - \lfloor x_j^{2k} \rfloor)$, $j \in N^1$, or $(y_j^{2k} - \lfloor y_j^{2k} \rfloor)$, $j \in N^2$, is the largest. Suppose x_j^2 , $j \in N^1$, an integer variable controlled by the leader, is selected as the branching variable. This means a new bound must be placed on this variable. If the selected variable's coefficient in the leader's objective functions is positive, a lower bound is placed on x_j^2 . The new lower bound is $x_j^2 \geq \lfloor x_j^{2k} \rfloor + 1$. If the coefficient is negative, an upper bound is placed on x_j^2 . The new upper bound is $x_j^2 \leq \lfloor x_j^{2k} \rfloor$. The vertex counter, k , is

incremented by 1, the bound vector, either α^{1k} or β^{1k} , is updated, and the index set of restricted integer variables, S_k^1 , is updated. The branching is designed to make the variables go in a direction which will provide increases in the value of the leader's objective function. A similar procedure is followed if the selected variable is controlled by the follower. This adds a live vertex to the branch and bound tree. Live vertices are added in a depth wise manner. The algorithm then returns to Step 1. If no integer variable has a fractional value, then the integer requirements of problem (6.1) are satisfied, and the algorithm proceeds to Step 4.

At Step 4, a solution in the inducible region of the mixed integer BLPP is found. The variables controlled by the leader are fixed at their value in the solution found at Step 2, and the follower's problem is solved. The follower's problem is a mixed integer programming problem where x^1 and x^2 are fixed. Its form is shown in (6.6).

Since the solution $(\tilde{x}^k, \tilde{y}^k)$ satisfies the constraints of (6.6), the mixed integer programming problem solved at Step 4 must have a feasible solution. The solution of

$$\text{Max}_{\mathbf{y}} f(\mathbf{y}) = d^{21}y^1 + d^{22}y^2 \quad (6.6a)$$

$$\text{subject to} \quad B^1y^1 + B^2y^2 \leq b - A^1x^1 - A^2x^2 \quad (6.6b)$$

$$y^1, y^2 \geq 0 \quad (6.6c)$$

$$y^2 \text{ integer} \quad (6.6d)$$

(6.6), (\hat{x}^k, \hat{y}^k) , is a solution of (6.1), and if it is greater than the current incumbent solution, it becomes the incumbent solution. As shown earlier in Examples 6.1 and 6.2, fathoming cannot be performed when a solution satisfying the integer requirements has been found, so the algorithm goes to Step 5.

At Step 5, an integer variable is selected for branching. The integer variable selected is one whose upper and lower bounds at vertex k are not equal to each other. Since the integer requirements of (6.1) are satisfied when the algorithm reaches Step 5, determining the variable on which to branch is somewhat arbitrary. The method used by the algorithm involves the use of a list. The first integer variable in the list whose upper and lower bounds at vertex k are not equal is selected as the branching variable. The selected

variable is then placed at the bottom of the list. The reason for this approach is explained in the section on computational results. Once a variable is selected, a new bound is placed on it. The method used is the same as the one used in Step 3. If the variable's coefficient in the leader's objective function is positive, a lower bound is placed on the variable, and if its coefficient is negative, an upper bound is placed on the variable. Also, the updating performed at Step 5 is the same as that done at Step 3. Again, a live vertex is added to the branch and bound tree in a depth wise fashion. Once a variable has been selected for branching and the updating has been accomplished, the algorithm returns to Step 1. If all the integer variables are fixed in the branch and bound tree, the algorithm goes to Step 6.

At Step 6, the algorithm backtracks. Once again, to facilitate bookkeeping, the path P_k in the branch and bound tree can be represented by an l -dimensional vector where l is the current depth of the tree. The order of the components is determined by their "level" in the tree. Indices only appear in the vector P_k if they are

in S_k^1 or S_k^2 . If $j \in S_k^1$ or $j \in S_k^2$, let it appear in P_k as

- j if $j \in S_k^1$ and variable j has an upper bound;
- $-j$ if $j \in S_k^1$ and variable j has a lower bound;
- \hat{j} if $j \in S_k^2$ and variable j has an upper bound;
- $-\hat{j}$ if $j \in S_k^2$ and variable j has a lower bound.

If the alternative bound has been considered, the entry in P_k appears underlined. In backtracking, the rightmost nonunderlined entry is underlined, its sign is changed, and all entries to its right are erased. If this procedure is followed, no integer combination can be overlooked in the backtrack operation. At Step 6, the backtracking is accompanied by updates of k , H_k^1 , H_k^2 , S_k^1 , and S_k^2 . The vertex counter k is incremented by 1, the bounds of H_k^1 and H_k^2 are updated as necessary, and the indexes of restricted variables, S_k^1 and S_k^2 , are added to and subtracted from depending on which variables continue to be restricted at the new vertex of the tree. It is instructive to note that a variable may appear in P_k many times. For example, if the problem has one integer variable controlled by the leader, this variable may have a lower bound of five at vertex 7 and

a lower bound of six at vertex 11. Thus, this variable's index would appear in P_{11} twice.

At Step 7, the algorithm terminates. If $\underline{F} = -\infty$, the mixed integer BLPP has no feasible solution. If $\underline{F} \neq -\infty$, a solution of the mixed integer BLPP has been found. At this point, the optimality of the solution found must be addressed. If variables controlled by the leader are restricted to integer values, the basic algorithm finds the optimal solution of the mixed integer BLPP. This is formally stated in the following proposition.

Proposition 6.2: If all the variables controlled by the leader are restricted to integer values, then under the uniqueness assumption stated in Chapter 3, the algorithm finds the optimal solution of (6.1).

Proof: Steps 3 and 5 ensure all possible combinations of x are implicitly considered; the subproblem solved at Step 4 ensures that all incumbent solutions are in the inducible region of (6.1). ■

If the variables controlled by the follower are continuous, then the basic algorithm will find the

optimal solution of the mixed integer BLPP when additional fathoming rules are applied. Proposition 6.3 formally states this.

Proposition 6.3: If all the variables controlled by the follower are continuous, then under the uniqueness assumption of Chapter 3, the basic algorithm will find the optimal solution of problem (6.1) if fathoming types 2 (relaxed solution \geq incumbent solution of the mixed integer BLPP) and 3 (solution satisfies the integrality requirements) are applied.

Proof: Each relaxed subproblem takes the form of problem (5.1). By Proposition 5.1, the algorithm used to solve the relaxed subproblem terminates with the optimal solution of the relaxed subproblem. Since all variables controlled by the follower are continuous, no restrictions are placed on them in the subproblems created by the basic algorithm. Hence, all solutions of the relaxed subproblems which satisfy the integrality requirements are in the inducible region of the mixed integer BLPP. Using fathoming types 2 and 3 will not prevent the basic algorithm from uncovering the optimal

solution of the mixed integer BLPP since placing more restrictions on the integer valued variables controlled by the leader and continuing to branch will only further restrict the leader's decisions. So, each solution which satisfies the integrality requirements is in the inducible region and applying fathoming types 2 and 3 does not preclude finding the optimal solution. Thus, the optimal solution of the mixed integer BLPP is uncovered. ■

Unfortunately, the basic algorithm may not produce an optimal solution when at least one of the variables controlled by the leader is continuous and one of the variables controlled by the follower is integer. The following example illustrates this.

Example 6.4

$$\begin{array}{ll} \text{Max } F(x,y) = x + 10y & \text{where } y \text{ solves} \\ x & \end{array}$$

$$\begin{array}{l} \text{Max } f(x,y) = -y \\ y \end{array}$$

$$\text{subject to } -25x + 20y \leq 30$$

$$x + 2y \leq 10$$

$$2x - y \leq 15$$

$$2x + 10y \geq 15$$

$$x, y \geq 0$$

$$y \text{ integer}$$

Example 6.4 is identical to Example 6.1 except that the variable controlled by the leader is continuous. The optimal solution of Example 6.1 is $(x^*, y^*) = (2, 2)$ with $F = 22$. The optimal solution of Example 6.4 is $(x^*, y^*) = (2.5 - \epsilon, 2)$ where $\epsilon > 0$ is arbitrarily small. The value of the leader's objective function at this point is $F = 22.5 - \epsilon$. If the leader chooses $x = 2.5$, the follower's decision is $y = 1$, and $F = 12.5$. Theorem 3.2 says that the solution of the linear BLPP is at an extreme point of the BLPP constraint region. The relaxed problem solved at Step 2 is a linear BLPP. Unfortunately, the bounds placed on y at Steps 3 and 5 never create a linear BLPP which has an extreme point at the optimal value of x in Example 6.4. The best feasible solution found by the algorithm is $(x, y) = (2, 2)$. This solution is found when the constraint $y \geq 4$ is added. If $y \geq 4$, the solution of the relaxed BLPP is $(x, y) = (2, 4)$. When x is fixed at 2 at Step 4 and the follower's problem is solved, the solution is $(x, y) = (2, 2)$. The following proposition formally states the fact that the algorithm

does not guarantee an optimal solution of (6.1).

Proposition 6.4: If, in (6.1), one or more of the variables controlled by the leader is real valued and one or more of the variables controlled by follower is integer valued, the algorithm finds a feasible but not necessarily optimal solution of (6.1).

6.6.2 Development of Heuristics

As shown in Example 6.4, a complete enumeration of the integer variables of (6.1) does not guarantee discovery of an optimal solution. Thus, the algorithm is a heuristic which, as shown in Proposition 6.4, finds a solution in the inducible region of (6.1). Unfortunately, for problems with 10 or more integer variables, the algorithm may require almost complete enumeration of the integer valued variables before terminating with a solution in the inducible region of (6.1). The efficiency of the algorithm can be improved by including an additional bound and fathoming rules. These changes weaken the algorithm and make it less likely that the true optimum will be found. The additional bound and fathoming rules represent a

tradeoff between accuracy and computational effort. At Step 2, a bound on the solution of the relaxed problem is considered, and at Step 4, three different fathoming rules can be included. A combination of these changes yields eight different algorithms which must be explored.

A bound which can be easily incorporated in the algorithm is based on the solution of the relaxed BLPP found at Step 2. When this solution gives a value for the leader's objective function which is less than the incumbent solution, fathoming is accomplished. This is analogous to fathoming type 2 in Section 6.2. The change is shown in Step 2a.

Step 2a: (Relaxed Solution) Attempt to solve the relaxed BLPP (6.2). If infeasible, go to Step 6. If successful, label the solution $(\underline{x}^k, \underline{y}^k)$. Compute $F(\underline{x}^k, \underline{y}^k)$. If $F(\underline{x}^k, \underline{y}^k) \leq \underline{F}$, go to Step 6; otherwise go to Step 3.

Several fathoming rules can be applied at Step 4. Three rules are selected for study and are shown below in Steps 4a, 4b, and 4c.

Step 4a: (Feasible Point) Fix x at \bar{x}^k and solve the follower's problem to obtain (\bar{x}^k, \hat{y}^k) . Compute $F(\bar{x}^k, \hat{y}^k)$. If $\underline{F} < F(\bar{x}^k, \hat{y}^k)$, put $\underline{F} \leftarrow F(\bar{x}^k, \hat{y}^k)$ and go to Step 5; otherwise go to Step 6.

Step 4b: (Feasible Point) Fix x at \bar{x}^k and solve the follower's problem to obtain (\bar{x}^k, \hat{y}^k) . Compute $F(\bar{x}^k, \hat{y}^k)$ and put $\underline{F} = \max[\underline{F}, F(\bar{x}^k, \hat{y}^k)]$. If $\hat{y}^k = \bar{y}^k$, go to Step 6; otherwise go to Step 5.

Step 4c: (Feasible Point) Fix x at \bar{x}^k and solve the follower's problem to obtain (\bar{x}^k, \hat{y}^k) . Compute $F(\bar{x}^k, \hat{y}^k)$ and put $\underline{F} = \max[\underline{F}, F(\bar{x}^k, \hat{y}^k)]$. Go to Step 6.

Step 4a allows the algorithm to go to Step 6 and backtrack when the solution at the current vertex of the branch and bound tree satisfies the integer requirements and the point in the inducible region derived from this solution does not have a value greater than that of the incumbent solution of (6.1). Step 4b sends the algorithm to Step 6 when the solution at the current

vertex satisfies the integer requirements of (6.1) and is in the inducible region of the mixed integer BLPP. Step 4c allows the algorithm to backtrack at Step 6 when the integer requirements of (6.1) are satisfied. Using various combinations of the above steps, seven additional algorithms which find feasible solutions of (6.1) are formed. The eight algorithms are shown in Table 6.1.

TABLE 6.1: THE EIGHT ALGORITHMS

Algorithm	Description
1	The basic algorithm,
2	Algorithm 1 with Step 4 replaced by Step 4a,
3	Algorithm 1 with Step 4 replaced by Step 4b,
4	Algorithm 1 with Step 4 replaced by Step 4c,
5	Algorithm 1 with Step 2 replaced by Step 2a,
6	Algorithm 5 with Step 4 replaced by Step 4a,
7	Algorithm 5 with Step 4 replaced by Step 4b,
8	Algorithm 5 with Step 4 replaced by Step 4c.

The next section presents a demonstration of the algorithms.

6.6.3 Demonstration

Example 6.2 is used to demonstrate how the algorithms work. The discussion focuses on Algorithm 1 and shows where the other algorithms would backtrack.

Example 6.2 is

Example 6.2

$$\begin{array}{ll} \text{Max } F(x,y) = -x - 2y & \text{where } y \text{ solves} \\ x & \end{array}$$

$$\begin{array}{l} \text{Max } f(x,y) = y \\ y \end{array}$$

$$\text{subject to } -x + 2.5y \leq 3.75$$

$$x + 2.5y \geq 3.75$$

$$2.5x + y \leq 8.75$$

$$x, y \geq 0$$

$$x, y \text{ integer.}$$

A graph of this example is shown in Figure 6.2. Because the variable controlled by the leader is integer, Algorithm 1 finds the optimal solution of this problem. To shorten the discussion, the updates of H_k^1 , H_k^2 , S_k^1 , and S_k^2 are not given. Rather, the bounds on x and y and the path vector P_k at each vertex are given. For convenience, x and y are labeled 1 and 2, respectively, in the path vector. The branch and bound tree created in solving Example 6.2 has 18 vertices and is shown in Figure 6.6. At vertex zero, Algorithm 1 solves the high point problem. At Step 2, the solution of the relaxed BLPP is $(\bar{x}^0, \bar{y}^0) = (0, 1.5)$ with $F = -3$. At Step 3, the

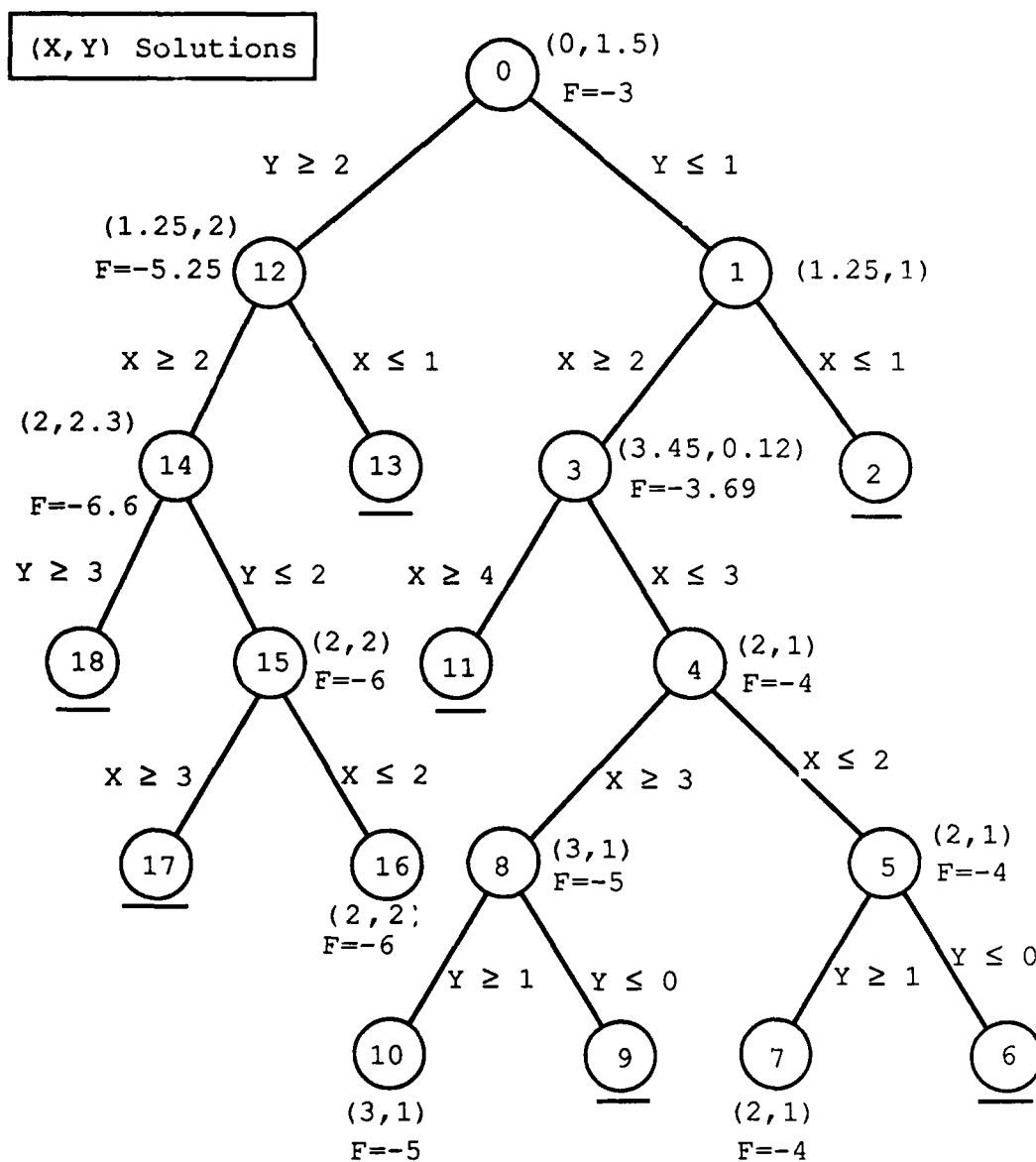


Figure 6.6: Branch and Bound Tree for Example 6.2

algorithm places an upper bound of $y \leq 1$. Thus, at vertex 1, $0 \leq x \leq \infty$, $0 \leq y \leq 1$ and $P_1 = \{2\}$. Returning to Step 1, the high point solution does not provide a bound, so the algorithm solves the relaxed BLPP at Step 2 with $(\underline{x}^1, \underline{y}^1) = (1.25, 1.0)$ and $F = -3.25$. At Step 3, x is the branching variable, and the bounds are $0 \leq x \leq 1$ and $0 \leq y \leq 1$ with $P_2 = \{2, 1\}$. The resulting BLPP is infeasible at Step 2, so the algorithm goes to Step 6 and backtracks. The new bounds are $0 \leq y \leq 1$ and $2 \leq x \leq \infty$ with $P_3 = \{2, -1\}$. Returning to Step 1, the high point problem is again solved. At Step 2, the solution of the relaxed BLPP is $F = -3.69$ with $(\underline{x}^3, \underline{y}^3) = (3.45, 0.12)$. At Step 3, an upper bound of 3 is placed on x , and the bounds on the variables are $2 \leq x \leq 3$ and $0 \leq y \leq 1$ with $P_4 = \{2, -1, 1\}$. Going to Step 1, backtracking cannot be done, so at Step 2, the relaxed BLPP's solution at vertex 4 is $(\underline{x}^4, \underline{y}^4) = (2, 1)$ with $F = -4$. At Step 3, the integer requirements are satisfied, so the algorithm proceeds to Step 4. The variable x is fixed at 2, and the follower's problem is solved to get the point $(\underline{x}^4, \hat{\underline{y}}^4) = (2, 2)$ with $F = -6$. The incumbent is updated, so $\underline{F} = -6$. At Step 5, the variable x is at the top of the list, so it is the

branching variable. A new upper bound is placed on x , so the bounds on the variables are $2 \leq x \leq 2$ and $0 \leq y \leq 1$ with $P_5 = \{2, -1, 1, 1\}$. Returning to Step 1, the current vertex cannot be fathomed. At Step 2, the solution is $(\underline{x}^5, \underline{y}^5) = (2, 1)$ with $F = -4$. At Step 3, the integer requirement is satisfied, so the algorithm drops to Step 4. Fixing x at 2, the solution of the follower's problem is $y = 2$, so $(\underline{x}^5, \hat{\underline{y}}^5) = (2, 2)$ and $F = -6$. This is not an improvement on the incumbent solution, so updating is not done. At Step 5, y is the only variable on which branching can be done. An upper bound of 0 is placed on y , so the bounds on the variables are $2 \leq x \leq 2$ and $0 \leq y \leq 0$. $P_6 = \{2, -1, 1, 1, 2\}$, and going to Step 1, the algorithm cannot backtrack. At Step 2, the relaxed BLPP is infeasible, so the algorithm proceeds to Step 6, backtracks with $P_7 = \{2, -1, 1, 1, -2\}$ as a lower bound of 1 is placed on y , and returns to Step 1. The bounds on the variables are now $2 \leq x \leq 2$ and $1 \leq y \leq 1$. At vertex 7, Step 1 does not allow backtracking, and the solution at Step 2 is $(\underline{x}^7, \underline{y}^7) = (2, 1)$. Because the solution satisfies the integer requirements, Step 3 sends the algorithm to Step 4. At Step 4, x is fixed at 2 and $(\underline{x}^7, \hat{\underline{y}}^7) = (2, 2)$ with

$F = -6$. The incumbent solution cannot be updated, and at Step 5, no variable can be branched on, so the algorithm goes to Step 6 where backtracking is done. A new lower bound is placed on x , so the bounds on the variables are $3 \leq x \leq 3$ and $0 \leq y \leq 1$ with $P_8 = \{2, \underline{-1}, 1, \underline{-1}\}$. Returning to Step 1, vertex 8 cannot be fathomed. At Step 2, the solution is $(\tilde{x}^8, \tilde{y}^8) = (3, 1)$ with $F = -5$. At Step 3, the integer requirement is satisfied, so the algorithm proceeds to Step 4. Fixing x at 3, the solution of the follower's problem is $y = 1$, so $(\tilde{x}^8, \hat{\tilde{y}}^8) = (3, 1)$ and $F = -5$. This is an improvement on the incumbent solution, and updating gives $\underline{F} = -5$. At Step 5, y is the only variable on which branching can be done. An upper bound of 0 is placed on y , so the bounds on the variables are $3 \leq x \leq 3$ and $0 \leq y \leq 0$. $P_9 = \{2, \underline{-1}, 1, \underline{-1}, 2\}$, and going to Step 1, the algorithm cannot backtrack. At Step 2, the relaxed BLPP at vertex 9 is infeasible, so backtracking is done. A lower bound of 1 is placed on y , and $P_{10} = \{2, \underline{-1}, 1, \underline{-1}, \underline{-2}\}$. Steps 2, 3, and 4 are the same as at vertex 8, and the algorithm reaches Step 5. Here, no variable can be branched on, so the algorithm goes to Step 6. Backtracking is done, and $P_{11} = \{2, \underline{-1}, \underline{-1}\}$. The bounds on the variables are

$4 \leq x \leq \infty$ and $0 \leq y \leq 1$. The algorithm then continues at Step 1. Algorithm 1 continues on to termination and creates a branch and bound tree with 18 vertices. The optimal solution is $(x^*, y^*) = (3, 1)$ with $F = -5$ and was discovered at vertex 8.

Algorithms 2, 3, 5, 6, and 7 also found the optimal solution of Example 6.2. Because Algorithms 4 and 8 backtrack when they find an integer solution, they did not find the optimal solution. The solution in the inducible region Algorithms 4 and 8 found was $(x, y) = (2, 2)$ with $F = -6$, and it was uncovered at vertex 4. The information on the trees produced by Algorithms 1 through 8 is presented in Table 6.2. The vertices which appear in the branch and bound trees created by the eight algorithms are listed in the table. The vertices listed correspond to those shown in Figure 6.6.

TABLE 6.2: BRANCH AND BOUND TREE DATA FOR EXAMPLE 6.2

Algor	No. of Vertices	Vertices in Branch and Bound Tree
1	18	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18
2	16	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-18
3	14	1-2-3-4-5-6-7-8-11-12-13-14-15-18
4	10	1-2-3-4-11-12-13-14-15-18
5	12	1-2-3-4-5-6-7-8-9-10-11-12
6	12	1-2-3-4-5-6-7-8-9-10-11-12
7	10	1-2-3-4-5-6-7-8-11-12
8	8	1-2-3-4-11-12-13-14

The tree for Algorithm 2 does not contain vertices 16 and 17. These vertices are not explored because at vertex 15, an integer solution of the relaxed BLPP is found which does not provide an improvement over the incumbent solution, so the algorithm backtracks. The tree for Algorithm 3 does not contain vertices 9, 10, 16, and 17. This happens because at vertices 8 and 15, solutions of the relaxed BLPP are integer and in the inducible region of Example 6.2. This also explains why vertices 9 and 10 are not explored by Algorithm 7. Algorithm 4 does not explore vertices 5, 6, 7, 8, 9, 10, 16, and 17. The reason for this is that at vertices 4 and 15, the relaxed BLPP has an integer solution. This also explains why Algorithm 8 does not inspect vertices

5 through 10. Algorithm 5 does not include vertices 13 through 18 in its branch and bound tree. This is because the solution of the relaxed BLPP at vertex 12 is less than the incumbent solution. For this reason, Algorithms 6 and 7 do not explore vertices 13 to 18. Algorithm 8 does not inspect vertices 15 to 18. This happens because the solution of the relaxed BLPP at vertex 14 is less than the current incumbent solution.

6.7 Computational Results

The results presented in this section show the outcomes of experiments used to determine the method for selecting the branching variable at Steps 3 and 5 of the algorithm. Ten sample problems are randomly generated and used to test various rules for selecting the branching variable at Steps 3 and 5. This section also presents the results achieved using the eight algorithms to find feasible solutions of randomly generated mixed integer BLPPs.

The problems used to test the various rules at Steps 3 and 5 and the eight algorithms were created using the same random problem generator. In each randomly generated problem, the coefficients of the A^1 , A^2 , B^1 , and B^2 matrices range between -15 and 45 with

approximately 25 percent of the entries being negative. The density of each matrix is between 0.4 and 0.5 and averages 0.47. The coefficients of the two objective functions vary between -25 and 25 with approximately 50 percent less than zero. The number of constraints in each problem is set at 0.4 times the total number of variables, and the right-hand side values range between 0 and 50. The signs of the constraints had a 0.7 probability of being \leq and a 0.3 probability of being \geq . The number of integer variables controlled by the leader and follower are set, and the variables flagged as integer are randomly selected.

The linear programming subroutine library XMP (Marsten 1981) is used to solve the linear problems encountered at Steps 1, 2, and 4 of the algorithms. In creating the bounds on the integer variables, constraints are not added to the original constraint set. The XMP subroutines can be used to solve a bounded variable linear program. This feature permits the use of upper and lower bounds on the variables. In the code which computerizes the algorithms, the bounds on each variable are maintained and adjusted as the algorithms place and reset the bounds on each variable. These

bounds are then included in the problem solved by the XMP subroutines.

At Step 4, the algorithm solves a mixed integer programming problem. The variables controlled by the leader are fixed, and the follower's problem (6.6) is solved. Utilizing XMP, the basis associated with the current point is used as the starting basis. The constraints of the problem do not have to be altered in this approach. Since the variables controlled by the leader are fixed, the XMP subroutines will work with the constraints in their original form. The coefficients of the variables in the objective function are given the values they have in the follower's objective function. The only bounds on the variables controlled by the follower are those of the mixed integer BLPP (6.1). All restrictions on the slack variables and Kuhn-Tucker multipliers are relaxed. Once these changes have been made, a branch and bound approach is used to solve the follower's problem. The XMP subroutines are used to do this, and bounds on the integer valued variables controlled by the follower are added by using the upper and lower variable bounds maintained in the XMP subroutines.

6.7.1 Determining a Branching Variable Selection Rule

At Step 3, 11 different rules for selecting the integer variable on which to branch are explored. The rules are listed below.

1. Branch on the integer variable with the largest fractional part.
2. Branch on the integer variable with the smallest nonzero fractional part.
3. Branch on the integer variable which is not integer valued and has the largest product for the variable value times its contribution to the leader's objective function.
4. Branch on the integer variable which is not integer valued and has the smallest product for the variable value times its contribution to the leader's objective function.
5. Branch on the integer variable which is not integer valued and has the largest value.
6. Branch on the integer variable which is not integer valued and has the smallest value.

7. Branch on the leader's variables before the follower's variables using rule 1 to choose the branching variable.
8. Branch on the follower's variables before the leader's variables using rule 1 to choose the branching variable.
9. Branch on the integer variable which is not integer valued and has the largest coefficient in the leader's objective function.
10. Branch on the integer variable which is not integer valued and has the smallest coefficient in the leader's objective function.
11. Branch on the integer variable whose fractional part is closest to 0.5.

These rules are tested using Algorithm 7 because the CPU time it requires to solve the test problems is, in general, reasonable for the test problems used to evaluate the above rules. The 10 test problems are of various size as shown in Table 6.3.

TABLE 6.3: TEST PROBLEM SIZE

Prob. No.	No. of Variables ($n^1 + n^2$)	Follower Variables (n^2)	Integer Leader Variables ($n^{1,2}$)	Integer Follower Variables ($n^{2,2}$)
1	12	5	4	1
2	15	5	5	0
3	20	10	4	3
4	25	10	5	5
5	25	10	6	4
6	25	10	4	6
7	25	10	7	3
8	25	10	7	3
9	25	10	10	0
10	35	15	4	5

In testing the rules, a 300 second CPU time limit was used, and if this limit was exceeded, the algorithm was halted. Results were achieved using an IBM 3081-D computer. Rule 1 provided the best results. The average CPU time required to solve the 10 test problems when rule 1 was used at Step 3 of Algorithm 7 was less than the average CPU time required when the other rules were applied. The results are interesting and are presented in Table 6.4. Table 6.4 also shows the results achieved in testing the rules at Step 5.

TABLE 6.4: CPU TIME (SECS) FOR RULES AT STEPS 3 AND 5

Step	Rule	Problem									
		1	2	3	4	5	6	7	8	9	10
3	1	3	1	2	72	6	16	142	5	23	67
3	2	4	2	2	109	9	20	188	7	31	117
3	3	4	2	3	127	7	22	†	8	34	†
3	4	4	1	1	107	8	17	166	7	27	95
3	5	2	2	1	77	7	19	164	6	29	106
3	6	4	2	2	108	6	17	155	8	27	77
3	7	3	1	2	101	7	22	173	7	26	106
3	8	3	2	2	80	6	18	155	6	28	110
3	9	3	1	3	124	8	24	182	8	25	89
3	10	2	2	1	103	7	19	151	7	30	94
3	11	4	2	3	137	9	21	†	7	33	†
5	1	3	1	2	63	8	27	162	5	19	87
5	2	3	1	2	77	9	15	178	6	21	96
5	3	3	1	2	72	6	16	142	5	23	67

†CPU time exceeded 300 seconds

Averages could not be computed for rules 3 and 11. While attempting to solve problems 7 and 10, the algorithm was halted because the CPU time exceeded 300 seconds. For problems 1, 2, and 3, the CPU time varied among the various rules by no more than 0.8 seconds. The longest required CPU time for these three problems was 4.2 seconds. Rule 6 was the second best rule. Its average CPU time exceeded rule 1's by 7 seconds. Ordering the remaining rules from best to worst gives 8,

5, 10, 4, 7, 9, and 2. Three problems really affected the outcome of the averaging. They are problems 4, 7, and 10. The greatest difference observed is 51 seconds. This occurred on problem 4 between rules 1 and 9. Once rule 1 was selected as the rule for choosing the branching variable, the direction of the branching had to be determined. Three rules were considered. The rules are always placing a lower bound on the variable, always placing an upper bound on the variable, or letting the sign of the variable's coefficient in the leader's objective function determine whether a lower or upper bound is placed on the variable. For seven of the 10 test problems, always placing an upper bound on the variable required between 2 and 6 additional seconds CPU time, and this represents a 15 percent increase in the average CPU time for these seven problems. The two other rules' CPU times are much closer. On three of the problems, the two bounding rules required the same amount of CPU time. On three other problems, always placing a lower bound on the variable required less CPU time. The improvement over basing the bound on the variable's coefficient in the leader's objective function is 0.5 to 1.0 seconds which is an eight percent

difference. On the four remaining problems, basing the bound on the variable's coefficient in the leader's objective function provided the best results. The range of the improvement is 0.5 to 3.0 seconds or a 12 percent improvement. These improvements are achieved on the more difficult problems which are problems 4, 7, and 10. Because of this, the bound placed on the variable selected for branching is based on the variable's coefficient in the leader's objective function. If the coefficient is zero or positive, a lower bound is placed on the variable, and if the coefficient is negative, an upper bound is placed on the variable.

At Step 5, devising a rule for selecting a branching variable when all the integer variables have integer values in the current solution is difficult. Three rules for doing this are tested. The rules are

1. Branch on a variable whose upper and lower bounds are not equal and whose level in the tree along the current path is nearest vertex zero.
2. Branch on a variable whose upper and lower bounds are not equal and whose level in the tree along the current path is farthest from vertex zero.

3. Branch on a variable whose upper and lower bounds are not equal and whose position in a list is nearest the top. When a variable is selected in this manner, the variable is placed at the bottom of the list. The list is initialized with the variables ordered according to their coefficient in the leader's objective function. The order is from largest to smallest.

For each rule, the direction of bounding is determined by the selected variable's value in the leader's objective function. For each rule, if the variable's coefficient in the leader's objective function is negative, an upper bound is placed on the variable, and if the coefficient is positive, a lower bound is placed on the variable. The results, which are presented in Table 6.4, show Rules 1 and 2 require more CPU time than Rule 3. The poor results of Rules 1 and 2 occurred when the same variable was branched on many times before the algorithm backtracked. For the 10 problems which are solved using the three rules, the average CPU time using Rule 3 is four seconds or 10 percent less than the average time achieved by rule 1 and seven seconds or 18

percent less than the average achieved by Rule 2. Thus, Rule 3 is selected as the branching rule at Step 5.

6.7.2 Testing the Eight Algorithms

To test the eight algorithms, 50 problems are randomly generated and solved using an IBM 3084-DQX computer. Ten classes of problems with five problems in each class are generated. The information on these problems is shown in Table 6.5.

TABLE 6.5: TEST PROBLEM SIZE

Class No.	No. of Variables ($n^1 + n^2$)	Follower Variables (n^2)	Integer Leader Variables (n^{1*2})	Integer Follower Variables (n^{2*2})
1	15	5	2	3
2	15	5	3	4
3	20	10	5	5
4	25	10	5	5
5	30	15	5	5
6	30	15	7	8
7	35	15	5	5
8	35	15	8	9
9	40	20	5	5
10	40	20	10	10

Tables 6.6, 6.7, and 6.8 summarize the computational results achieved using the eight algorithms. Results are shown as averages for each problem class. Table 6.6 shows the average CPU time

required by each algorithm to reach termination. Table 6.7 displays the average number of vertices needed by each algorithm to reach termination. Table 6.8 shows the average number of vertices required by each algorithm to find the best solution in the inducible region an algorithm can uncover.

TABLE 6.6: AVERAGE CPU TIME FOR ALGORITHMS

Problem Class	Alg. 1	Alg. 2	Alg. 3	Alg. 4	Alg. 5	Alg. 6	Alg. 7	Alg. 8
1	5	1	3	1	3	1	3	1
2	210	1	1	1	1	1	1	1
3	186	27	105	17	49	12	35	9
4	73	5	5	4	6	5	5	4
5	678	77	837	70	683	57	730	50
6	244	46	113	41	99	36	97	32
7	170	42	33	28	41	38	28	24
8	634	178	422	175	89	72	84	66
9	366	103	235	81	118	78	107	59
10	730	272	454	311	123	123	118	118

TABLE 6.7: AVERAGE NUMBER OF VERTICES IN THE BRANCH AND BOUND TREE

Problem Class	Alg. 1	Alg. 2	Alg. 3	Alg. 4	Alg. 5	Alg. 6	Alg. 7	Alg. 8
1	86	21	56	18	51	21	49	18
2	250	12	10	8	14	12	10	8
3	541	60	313	54	169	32	166	26
4	243	11	11	8	12	10	10	7
5	483	49	415	42	343	26	362	20
6	380	40	239	35	225	27	223	24
7	90	15	15	11	13	11	9	6
8	366	115	249	117	51	40	47	37
9	136	30	84	25	41	20	38	15
10	128	42	70	40	30	30	27	27

TABLE 6.8: AVERAGE NUMBER OF VERTICES TO FIND BEST FEASIBLE SOLUTION

Problem Class	Alg. 1	Alg. 2	Alg. 3	Alg. 4	Alg. 5	Alg. 6	Alg. 7	Alg. 8
1	60	19	56	18	49	19	48	18
2	44	10	10	8	12	10	10	8
3	212	33	201	28	148	27	146	23
4	10	7	6	6	7	7	6	6
5	78	26	60	22	62	20	57	16
6	216	23	216	21	209	22	208	21
7	59	8	8	5	10	8	8	5
8	94	90	36	96	46	34	44	34
9	107	26	74	23	38	17	37	14
10	36	20	27	20	20	20	20	20

In collecting the above results, several observations should be highlighted. Algorithm 1 had a tendency to get into a branching loop. That is, one particular variable would be selected for branching, and the

algorithm would be unable to find a vertex where it could backtrack. Since branching is done on variables which have integer values, a variable would be selected for branching, and as bounds were placed on the variable, the resulting subproblems would continue to be feasible. Because of the weak bound used to test a solution, Algorithm 1 is unable to backtrack, so many vertices can be added to the branch and bound tree before backtracking is accomplished. As a result of this, a restriction is placed on the number of vertices in the branch and bound tree. If the number of vertices reaches 995, the algorithm is terminated, and the incumbent solution is declared the best solution in the inducible region the algorithm could find. Algorithm 1 reaches the vertex limit on eight problems. Algorithm 3 exceeds the allowed number of vertices three times. Algorithms 5 and 7 each surpass the vertex limit twice. Algorithms 2, 4, 6, and 8 never exceed the vertex limit. Classes 1, 7, 9, and 10 have no problems which caused an algorithm to surpass the vertex limit.

On 11 of the 50 problems, the algorithms did not reach the same feasible solution. On 4 of the 11 disagreements, Algorithm 1 did not provide the best

feasible solution. In each case, the algorithm terminated because it exceeded the vertex limit. Algorithms 3, 5, and 7 each terminated once at the vertex limit without reaching a feasible solution as good as that found by one of the other algorithms. On three of the problems which the algorithms did not reach the same solution, vertex limit violation is the explanation. For the fourth problem, the best feasible solution was found by Algorithm 3 alone. On this particular problem, Algorithm 3 terminated because it had reached the vertex limit. Algorithms 1, 5, and 7 terminated at the vertex limit with solutions less than the one found by Algorithm 3. The other algorithms terminated normally without achieving the same solution as Algorithm 3.

For the other seven disagreements, Algorithm 1 alone found the best feasible solution three times. Of the four remaining disagreements, Algorithms 1, 2, 5, and 6 found the best solution twice, Algorithms 1, 3, 5, and 7 found the best solution once, and Algorithms 1, 2, 3, 5, 6, and 7 found the best solution once. These results are summarized in Table 6.9.

TABLE 6.9: NUMBER OF TIMES EACH ALGORITHM FAILS
TO FIND THE BEST FEASIBLE SOLUTION

Alg. 1	Alg. 2	Alg. 3	Alg. 4	Alg. 5	Alg. 6	Alg. 7	Alg. 8
4	5	6	8	4	5	6	8

The problem size does have some effect on the CPU time elapsed before termination is reached. The Class 5 problems illustrate a difficulty encountered in working with the mixed integer BLPP. Although the problems have only 30 variables, 10 of which are integer, the problems generated required as much CPU time as larger problems. This shows the problem dependency encountered when trying to solve the mixed integer BLPP. Depending on the problem generated, CPU time can be excessive or quite short. This problem dependency is starkly illustrated by two problems in Class 8. The difference in CPU time for two problems which did not abnormally terminate is 526 seconds. This difference is for Algorithm 1. For these same two problems, the difference in CPU time using Algorithm 8 is 123 seconds.

An examination of the data shows the algorithms can be placed in order based on average CPU time. Although this order does not always hold, it is usually true.

From fastest to slowest, the rank is 8, 6, 4, 2, 7, 5, 3, and 1. As noted above, some sacrifice must be made to decrease CPU time. Algorithms 4 and 8 failed to reach the best feasible solution found by one or more of the algorithms on eight of the 50 problems. These algorithms backtrack when a solution is found which satisfies the integrality requirements of the mixed integer BLPP. Similarly, Algorithms 2 and 6 failed to reach the best feasible solution on five of the 50 problems. They backtrack when no improvement is provided by a solution which satisfies the integrality requirements of the mixed integer BLPP. Algorithms 3 and 7 failed to find the best feasible solution on six of the problems, while Algorithm 5 failed on four of the 50 problems. Algorithms 3 and 7 backtrack when the solution of problem (6.2) is in the inducible region of the mixed integer BLPP problem, and Algorithm 5 backtracks when the solution of problem (6.2) is not better than the incumbent solution. The backtracking rule of Algorithm 5 is also employed in Algorithms 6, 7, and 8. If the best possible solution, regardless of CPU time, is desired, Algorithm 1 is the choice. However, if both the solution and CPU time are important,

Algorithm 2 offers a good compromise. If CPU time is an important concern, Algorithm 8 is the proper algorithm to use.

For a linear mixed integer bilevel programming problem, the algorithm which would best solve the problem is Algorithm 2. In testing, this algorithm never exceeded the vertex limit while solving a problem. It found the best feasible solution on 45 of the 50 problems, and in terms of CPU time, it ranked fourth. It offers a good balance between computational effort and ability to find a good solution of the mixed integer BLPP.

CHAPTER 7

THE BILEVEL PROGRAMMING PROBLEM AS APPLIED TO A PRODUCTION PLANNING PROBLEM

7.1 Introduction

This chapter centers on a production planning problem with inexact demand. This problem is modified by creating a customer who determines demand. One way of modeling the situation where a customer controls demand is as a bilevel programming problem (BLPP). Two variations of this formulation are investigated.

The production planning problem used as a base from which to develop the BLPP formulations comes from Bard and Chatterjee (1985). Their problem has a manufacturer of n products who wishes to determine a production mix that will maximize her profits over a finite time horizon, T , in the face of m resource constraints and an inexact demand forecast. The problem follows.

Problem 7.1

$$\begin{aligned} \text{Max}_{x, I, S, V} \quad & \sum_{j=1}^n \sum_{t=1}^T p_{jt} d_{jt} - \sum_{j=1}^n \sum_{t=1}^T [c_{jt} x_{jt} + h_{jt} I_{jt} \\ & + s_{jt} S_{jt} + v_{jt}] \end{aligned} \quad (7.1a)$$

subject to

$$d_{jt} - x_{jt} - I_{j,t-1} + I_{jt} - S_{jt} = 0 \quad \begin{matrix} j = 1, \dots, n \\ t = 1, \dots, T \end{matrix} \quad (7.1b)$$

$$\sum_{j=1}^n a_{ijt} x_{jt} \leq b_{it} \quad \begin{matrix} i = 1, \dots, m \\ t = 1, \dots, T \end{matrix} \quad (7.1c)$$

$$\sum_{j=1}^n v_{jt} \leq e_{it} \quad \begin{matrix} i = 1, \dots, l \\ t = 1, \dots, T \end{matrix} \quad (7.1d)$$

$$x_{jt} \geq 0, S_{jt} \geq 0, I_{jt} \geq 0, v_{jt} \geq 0 \quad \begin{matrix} j = 1, \dots, n \\ t = 1, \dots, T \end{matrix} \quad (7.1e)$$

$$d_t = (d_{1t}, \dots, d_{nt}) \in D_t \quad t = 1, \dots, T \quad (7.1f)$$

where

x_{jt} is the amount of product j manufactured in period t ;

a_{ijt} is the amount of resource i required to make product j in period t ;

b_{it} is the amount of resource i available in period t ;

- I_{jt} is the amount of inventory of product j in period t ;
- S_{jt} is the amount of shortage of product j in period t ;
- V_{jt} is the amount spent on advertising product j in period t ;
- e_{it} is the amount of money available in constraint i to spend on advertising in period t ;
- p_{jt} is the (known) selling price of product j in period t ;
- c_{jt} is the (known) unit cost of manufacturing product j in period t ;
- h_{jt} is the (known) unit holding cost of product j in period t ;
- s_{jt} is the (known) unit cost of subcontracting product j in period t ;
- d_{jt} is the (inexact) demand forecast for product j in period t ;
- D_t is the set in which the n -dimensional vector d_t lies in period t .

In the objective function (7.1a), the first term represents total revenues over the time horizon while

the second term sums the individual costs: production, inventory, shortage, and advertising. Equation (7.1b) is the material balance constraint, and (7.1c) defines the resource usage restrictions. Equation (7.1d) establishes the limits on advertising. To meet a demand which exceeds production in a period, the manufacturer has two options. She can meet the demand with inventory or by ordering additional product from a subcontractor. The manufacturer can also order excess product from a subcontractor and place the excess in inventory, so there is a tradeoff between subcontracting and holding inventory.

7.2 Formulation as a BLPP

If the demand d_{jt} is known exactly, then D_t is a singleton, and (7.1) reduces to a standard linear program. Now, if demand is treated as a random variable with some known distribution, a stochastic linear program is formed. Another way to approach this problem is to introduce a customer. In this instance, the problem can be reformulated as a BLPP if certain conditions are met. The manufacturer is given the role of the leader while the customer is cast in the role of the follower. To place the customer in the role of

follower, several assumptions are made. The first is that the customer knows the price of each product in each period of the time horizon. The second assumption is that the customer is rational. That is, the customer wants to minimize his costs which are the purchase prices he pays for products. The third assumption is that the customer does not make his purchase decisions until the manufacturer announces his production and advertising decisions for the time horizon. The last assumption is that the manufacturer wishes to satisfy demand in every period. As a consequence of the last assumption, the customer effectively controls inventory and shortages because once the manufacturer makes her production and advertising decisions, the customer determines the amount of inventory or shortage in each period through his purchasing decisions. The last assumption also has an impact on the formulation of the BLPP. The manufacturer's desire to satisfy all demand means that the shortages are, in reality, an amount of product which must be subcontracted. Since subcontracting can be done in each period, the manufacturer could order product from a subcontractor in a period, place this product in inventory, and use it to

meet shortages in a latter period. The manufacturer would do this if a future shortage could be predicted and the cost of subcontracting in a period prior to the predicted shortage plus the cost of inventory were less than the cost of subcontracting in the period for which the shortage is predicted. Now, in modeling this situation, a determination as to how to treat subcontracting must be made. If subcontracting is only allowed in periods when a shortage exists, this implies either inventory or shortage (or both) of each product j in every period t is zero. If the manufacturer is allowed to meet a shortage in a period by subcontracting in a previous period, the possibility of multiple responses by the customer to the manufacturer's decision exists. This situation occurs because the formulation of the BLPP gives the customer control over product inventory and subcontracting. When a shortage arises, the customer can either select subcontracting in the current period or subcontracting and inventory in a previous period. Since neither choice has an effect on the customer's cost, the customer may or may not make a decision which allows the manufacturer to realize the largest profit possible from a given set of decisions.

It is difficult to determine which way to model the shortages. Only allowing subcontracting in the period in which shortages occur is realistic because the manufacturer does not know what demand is prior to the current period. However, subcontracting and placing the product in inventory allows the manufacturer to anticipate the customer's decision. To model the first situation requires the introduction of a quadratic penalty term in the customer's objective function. By penalizing the customer for having inventory and a shortage in the same period, subcontracting is allowed only in periods when a shortage arises. In the latter situation, the problem of multiple optima is overcome when the BLPP is recast in its equivalent Kuhn-Tucker formulation. In this formulation, the manufacturer's objective function (7.1a) becomes the objective function, and this means the occurrence of subcontracting product for inventory purposes will happen only when it is of benefit to the manufacturer.

The following problem is the BLPP derived from problem (7.1). Notice in (7.2b) that the customer's problem has been recast as a maximization problem to allow application of the algorithm developed in Chapter 5.

Problem 7.2

$$\begin{aligned} \text{Max}_{x,V} \quad & \sum_{j=1}^n \sum_{t=1}^T p_{jt} d_{jt} - \sum_{j=1}^n \sum_{t=1}^T [c_{jt} x_{jt} + h_{jt} I_{jt} \\ & + s_{jt} S_{jt} + v_{jt}] \end{aligned} \quad (7.2a)$$

where I , d , and S solve

$$\begin{aligned} \text{Max}_{I,d,S} \quad & \sum_{j=1}^n \sum_{t=1}^T -p_{jt} d_{jt} \end{aligned} \quad (7.2b)$$

subject to

$$d_{jt} - x_{jt} - I_{j,t-1} + I_{jt} - S_{jt} = 0 \quad \begin{matrix} j = 1, \dots, n \\ t = 1, \dots, T \end{matrix} \quad (7.2c)$$

$$\sum_{j=1}^n a_{ijt} x_{jt} \leq b_{it} \quad \begin{matrix} i = 1, \dots, m \\ t = 1, \dots, T \end{matrix} \quad (7.2d)$$

$$\sum_{j=1}^n v_{jt} \leq e_{it} \quad \begin{matrix} i = 1, \dots, l \\ t = 1, \dots, T \end{matrix} \quad (7.2e)$$

$$x_{jt} \geq 0, S_{jt} \geq 0, I_{jt} \geq 0, v_{jt} \geq 0 \quad \begin{matrix} j = 1, \dots, n \\ t = 1, \dots, T \end{matrix} \quad (7.2f)$$

$$d_t = (d_{1t}, \dots, d_{nt}) \in D_t \quad t = 1, \dots, T \quad (7.2g)$$

Problem (7.2) models the situation where the manufacturer is allowed to order product from a subcontractor and place it in storage for use in latter periods. The situation where the manufacturer can only order product from a subcontractor when shortages occur is modelled by changing the customer's objective

function, (7.2b). The modified objective function is

$$\text{Max}_{I,d,S} \sum_{j=1}^n \sum_{t=1}^T -p_{jt} d_{jt} - \sum_{j=1}^n \sum_{t=1}^T M S_{jt} I_{jt} \quad (7.2b)$$

where M represents a large number and the second term is a penalty for having both a shortage and inventory in the same period. The second term in the modified objective function is equivalent to adding the complementarity constraint $S_{jt} I_{jt} = 0 \forall j, t$ to the problem. One way to model these constraints in a linear problem is the introduction of binary variables. To avoid the introduction of integer variables, the complementarity constraint is added to the objective function as a penalty term.

The formulation of (7.2) allows the manufacturer to have an unlimited amount of inventory on hand. It also does not penalize her for any setup costs which she might incur. Problem (7.2) can be modified to include both setup costs and a limit on inventory. The addition of an upper bound on inventory would constrain the customer's decision. This happens when the customer wants to make a decision which would cause inventory to exceed its upper bound. An upper bound on inventory can

be included if the manufacturer rents extra storage space when her upper limit on storage space is reached. The modeling of this situation can be accomplished by using binary variables. Binary variables are also used to include the setup costs. The revised problem is

Problem 7.3

$$\begin{aligned} \text{Max}_{x,V,z} \quad & \sum_{j=1}^n \sum_{t=1}^T p_{jt} d_{jt} - \sum_{j=1}^n \sum_{t=1}^T [c_{jt} x_{jt} + h_{jt} I_{jt} \\ & + s_{jt} S_{jt} + v_{jt}] - \sum_{t=1}^T r_t y_t - \sum_{t=1}^T f_t z_t \end{aligned} \quad (7.3a)$$

where I , d , S , and y solve

$$\text{Max}_{I,d,y,S} \quad \sum_{j=1}^n \sum_{t=1}^T -p_{jt} d_{jt} \quad (7.3b)$$

subject to

$$d_{jt} - x_{jt} - I_{j,t-1} + I_{jt} - S_{jt} = 0 \quad \begin{matrix} j = 1, \dots, n \\ t = 1, \dots, T \end{matrix} \quad (7.3c)$$

$$\sum_{j=1}^n a_{ijt} x_{jt} \leq b_{it} \quad \begin{matrix} i = 1, \dots, m \\ t = 1, \dots, T \end{matrix} \quad (7.3d)$$

$$\sum_{j=1}^n v_{jt} \leq e_{it} \quad \begin{matrix} i = 1, \dots, l \\ t = 1, \dots, T \end{matrix} \quad (7.3e)$$

$$M y_t + T \text{MAX}_t - \sum_{j=1}^n B_j I_{jt} \geq 0 \quad t = 1, \dots, T \quad (7.3f)$$

$$M z_{jt} - x_{jt} \geq 0 \quad \begin{matrix} j = 1, \dots, n \\ t = 1, \dots, T \end{matrix} \quad (7.3g)$$

$$x_{jt} \geq 0, S_{jt} \geq 0, I_{jt} \geq 0, V_{jt} \geq 0 \quad \begin{matrix} j = 1, \dots, n \\ t = 1, \dots, T \end{matrix} \quad (7.3h)$$

$$y_t, z_t \in \{0,1\} \quad t = 1, \dots, T \quad (7.3i)$$

$$d_t = (d_{1t}, \dots, d_{nt}) \in D_t \quad t = 1, \dots, T \quad (7.3j)$$

where

M is a large number,

r_t is the cost of renting space in period t ;

f_t is the setup cost for production in period t ;

y_t is a binary variable which indicates if space is rented ($y_t=1$) or is not rented ($y_t=0$) in period t ;

z_t is binary variable which indicates if a setup cost is incurred ($z_t=1$) or not incurred ($z_t=0$) for production in period t ;

B_j is a relative measure of space occupied by product j and is normalized to product 1.

In (7.3), constraint (7.3f) represents the manufacturer's need to rent additional storage space, and (7.3g) is the constraint related to setup costs. In (7.3), the manufacturer pays a fixed fee to rent storage space, so she pays the same amount to rent space for one

item or 500 items. Problem (7.3) is a mixed integer problem, and its solution will require the use of the algorithms for the mixed integer bilevel programming problem developed in Chapter 6.

Notice that the setup costs are controlled by the manufacturer. Since these costs cannot be used to control the customer's decision making process, their control by the manufacturer is appropriate. However, the determination concerning renting additional storage space must be modelled as a customer decision. If control of this were given to the manufacturer, she could use it to constrain the customer's decision. The above formulation allows the manufacturer to meet shortages in a given period by ordering product from a subcontractor in prior periods. As in (7.2), if this situation is not allowed, the customer's objective function, (7.3b), can be modified to prevent this. This can be done by including a penalty term in (7.3b) which charges an unacceptable amount for having shortages and inventories of the same product in a period. The customer's revised objective function is presented below.

$$\text{Max}_{I,d,S} \sum_{j=1}^n \sum_{t=1}^T -p_{jt} d_{jt} - \sum_{j=1}^n \sum_{t=1}^T M S_{jt} I_{jt} \quad (7.3b)$$

Unfortunately, the algorithms for the mixed integer BLPP, as currently coded, cannot solve problems with quadratic terms in the follower's objective function. Rather, this situation can be modelled by using either-or constraints because either shortages or inventory of a product in a period must be zero. This requires the addition of constraints and binary variables to (7.3). The additional constraints are

$$S_{jt} - M w_{jt} \geq 0 \quad \begin{matrix} j = 1, \dots, n \\ t = 1, \dots, T \end{matrix} \quad (7.3k)$$

$$I_{jt} + M w_{jt} \geq M \quad \begin{matrix} j = 1, \dots, n \\ t = 1, \dots, T \end{matrix} \quad (7.3l)$$

$$w_{jt} \in \{0,1\}. \quad \begin{matrix} j = 1, \dots, n \\ t = 1, \dots, T \end{matrix} \quad (7.3m)$$

This formulation adds $n \times T$ binary variables and $2(n \times T)$ constraints to (7.3).

The formulations of (7.2) and (7.3) do not preclude the possibility of multiple optimal solutions to the customer's problem. Once the manufacturer has made her decisions, there are many ways for the customer to assign values to inventory and shortage which satisfy

the constraints and still allow the customer to minimize his costs. The assumption that the follower's rational reaction set, $\Psi(x^*)$, is a singleton for x^* optimal may not be true for this problem.

7.3 Example Problems

Constraints and data for production planning problems in the form of (7.2) and (7.3) were created to demonstrate and test the performance of the algorithms presented in Chapters 5 and 6. For each problem, the number of products is $n = 2$, the number of periods is $T = 4$, the number of resource constraints is $m = 6$, the number of constraints on advertising is $\ell = 7$, and the value of big M is 1500. The information and data on the problems are presented in the following tables.

TABLE 7.1: PROBLEM DATA

Coefficient	Period			
	1	2	3	4
p_{1t}	75	115	160	90
p_{2t}	115	200	300	160
c_{1t}	50	75	100	60
c_{2t}	75	95	200	100
h_{1t}	25	25	25	25
h_{2t}	25	25	25	25
s_{1t}	80	110	140	95
s_{2t}	110	170	260	140
f_t	7500	7500	7500	7500
r_t	3000	3000	3000	3000

Also, the value of I_{\max} is 50 in each period, $B_1 = 1$, and $B_2 = 1.5$. For the six resource constraints, the A matrix and right-hand side vector are presented below.

$$A = \begin{bmatrix} 3 & 3 & 3 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 2 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 \end{bmatrix}$$

and $b = [2750, 2000, 2700, 2700, 2700, 2700]$.

The manufacturer's marketing department has done an analysis of the advertising budget. The analysis shows the manufacturer should never spend more than \$10,000 on advertising and should never spend less than \$1,500 on advertising a product in two consecutive periods. These requirements generate seven constraints. The A matrix of these seven constraints is shown below. The first constraint is \leq and the other six are \geq .

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

and the right-hand side vector = [10000,1500,1500,1500,1500,1500,1500].

The analysis also reveals information about the relation between advertising and sales. The advertising scheme will guarantee a demand of 100 for each product. The demand for the products is also related to the amount of advertising done before and during a period. For each dollar spent on advertising in a period, a

certain amount of demand is generated. The following table presents this information.

TABLE 7.2: PRODUCT DEMAND PER DOLLAR OF ADVERTISING

Product 1				
Period in which Advertising done	Demand by Period			
	1	2	3	4
1	0.050	0.030	0.015	0.005
2	-----	0.035	0.020	0.010
3	-----	-----	0.0325	0.0175
4	-----	-----	-----	0.037

Product 2				
Period in which Advertising done	Demand by Period			
	1	2	3	4
1	0.080	0.020	0.015	0.005
2	-----	0.040	0.020	0.010
3	-----	-----	0.031	0.019
4	-----	-----	-----	0.039

The customer also has product requirements. In purchasing product 1, for each unit bought in period 1, two-thirds of a unit must be purchased in period 2; for each unit bought in period 2, one-half of a unit must be purchased in period 3; and for each unit purchased in period 3, three-fourths of a unit must be bought in period 4. In purchasing product 2, for each unit bought in period 1, one-half of a unit must be purchased in

period 2; for each unit bought in period 2, one-fourth of a unit must be purchased in period 3; and for each unit purchased in period 3, three-fifths of a unit must be bought in period 4. Using this information and the data presented in Table 7.2, the set D is formulated.

$$d_{11} + d_{12} + d_{13} + d_{14} \geq 100$$

$$d_{21} + d_{22} + d_{23} + d_{24} \geq 100$$

$$2d_{11} - 3d_{12} \leq 0$$

$$1d_{12} - 2d_{13} \leq 0$$

$$3d_{13} - 4d_{14} \leq 0$$

$$1d_{21} - 2d_{22} \leq 0$$

$$1d_{22} - 4d_{23} \leq 0$$

$$3d_{23} - 5d_{24} \leq 0$$

$$100d_{11} - 5V_{11} \geq 0$$

$$100d_{12} - 3V_{11} - 3.5V_{12} \geq 0$$

$$100d_{13} - 1.5V_{11} - 2V_{12} - 3.25V_{13} \geq 0$$

$$100d_{14} - 0.5V_{11} - 1V_{12} - 1.75V_{13} - 3.7V_{14} \geq 0$$

$$100d_{21} - 8V_{21} \geq 0$$

$$100d_{22} - 2V_{21} - 4V_{22} \geq 0$$

$$100d_{23} - 1.5V_{21} - 2V_{22} - 3.1V_{23} \geq 0$$

$$100d_{24} - 0.5V_{21} - 1V_{22} - 1.9V_{23} - 3.9V_{24} \geq 0$$

Problem (7.2) was solved for the case where the manufacturer is allowed to anticipate customer demand. For this problem, $n^1 = 16$, $n^2 = 24$, and the total number of constraints is 37. Although the manufacturer had the option to order product from a subcontractor and place it in inventory, the manufacturer did not take advantage of this option. In fact, the manufacturer's optimal solution does not order any product from a subcontractor. Therefore, it is not necessary to solve the modified version of problem (7.2) where the option of ordering product from a subcontractor and placing the product in storage is eliminated.

The algorithm of Chapter 5 was used to obtain the solution presented in Table 7.3. The manufacturer's profit for the results presented in Table 7.3 is \$68,317. The algorithm required 12.8 seconds of CPU time to solve the problem. The number of vertices the algorithm explored was 267, and the optimal solution was found at vertex 118. Since the manufacturer did not purchase any product from a subcontractor, data on subcontracting is omitted from Table 7.3.

TABLE 7.3: DEMAND, PRODUCTION, AND INVENTORY
SCHEDULE OF PROBLEM (7.2)

Variable	Period			
	1	2	3	4
Product 1 Production	75.0	121.25	0.0	53.4375
Product 2 Production	440.0	349.0	0.0	77.4
Product 1 Advertising	1500.0	0.0	1500.0	0.0
Product 2 Advertising	5500.0	0.0	1500.0	0.0
Product 1 Demand	75.0	50.0	71.25	53.4375
Product 2 Demand	440.0	220.0	129.0	77.4
Product 1 Inventory	0.0	71.25	0.0	0.0
Product 2 Inventory	0.0	129.0	0.0	0.0

Using the algorithms presented in Chapter 6 and solving problem (7.3) where the manufacturer can subcontract in any period provided some interesting results. For this problem, $n^1 = 20$ with four of the variables binary, $n^2 = 28$ with four of the variables binary, and the total number of constraints is 45. Seven of the algorithms arrived at a solution where the manufacturer made \$47,430, and one algorithm found a solution where the manufacturer's profit was \$45,351. The algorithm which failed to find the best result was Algorithm 8. The solution which yields a \$47,430 profit is shown in Table 7.4. As can be seen in Table 7.4, the manufacturer rented storage space in periods 1 and 2 and incurred setup costs in period 1 by manufacturing products 1 and 2. The manufacturer did order product from a subcontractor in period 4, but as in the solution to Problem (7.2), no product was bought from a subcontractor and placed in storage. This alleviates the need to solve the modified version of Problem (7.3) where the storage of subcontracted product is prohibited. Table 7.4 is shown below.

TABLE 7.4: DEMAND, PRODUCTION, INVENTORY, AND
SUBCONTRACTING SCHEDULE
OF PROBLEM (7.3)

Variable	Period			
	1	2	3	4
Product 1 Production	166.5	0.0	0.0	0.0
Product 2 Production	789.0	0.0	0.0	0.0
Product 1 Advertising	8812.5	6187.5	1500.0	0.0
Product 2 Advertising	5500.0	0.0	1500.0	0.0
Product 1 Demand	44.0625	48.094	74.344	55.758
Product 2 Demand	440.0	220.0	129.0	77.4
Product 1 Inventory	122.4375	74.344	0.0	0.0
Product 2 Inventory	349.0	129.0	0.0	0.0
Product 1 Subcontracted	0.0	0.0	0.0	55.758
Product 2 Subcontracted	0.0	0.0	0.0	77.4

The CPU time and number of vertices required by each algorithm to reach termination is given in Table 7.5. The vertex where each algorithm found the best solution is also shown.

TABLE 7.5: RESULTS OF ALGORITHMS FOR PROBLEM (7.3)

Algorithm	CPU Time (min)	Number of Vertices	Best Solution Vertex
1	862.98	510	355
2	63.28	70	47
3	411.83	252	175
4	57.52	66	43
5	332.60	194	121
6	59.83	64	42
7	323.03	182	113
8	44.71	54	53

Large CPU times were experienced because the algorithms had to solve many relaxed BLPPs at each vertex before reaching a solution which satisfied the integrality requirements. Since Algorithms 1 through 7 found the best solution for the problem, they can be rank ordered according to their CPU time. The rank order of the seven algorithms from fastest to slowest is 4, 6, 2, 7, 5, 3, and 1. In Chapter 6, the rank of Algorithms 4 and 6 is reversed. The rank order in Chapter 6 was based on average CPU time achieved solving 50 test problems.

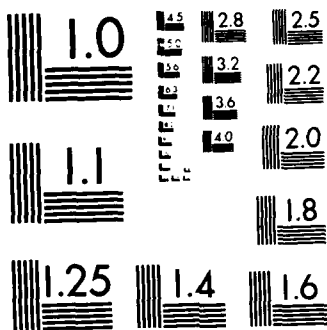
WD-ALSC 113

EXTENSIONS TO THE MULTILEVEL PROGRAMMING PROBLEM(U) AIR 3/4
FORCE INST OF TECH WRIGHT-PATTERSON AFB OH J T MOORE
MAY 88 AFIT/CI/MR-88-123

UNCLASSIFIED

F/G 12/5

ML



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

CHAPTER 8

CONCLUSIONS

8.1 Introduction

In the previous chapters, various versions of the bilevel programming problem (BLPP) have been examined. Algorithms have been developed which solve the BLPP with binary variables, with continuous variables, and with both integer and continuous variables. An application of BLPP to the production planning problem was also explored. Much remains to be done in developing applications of the BLPP and in formulating algorithms for the BLPP. The multilevel programming problem poses an even greater research challenge as much remains to be done in expanding the understanding and uses of it.

8.2 The 0-1 BLPP

After solving approximately 200 randomly generated problems, experience has shown that a wide variation in algorithmic performance exists, even for problems of the

same class. And while the number of variables greatly affects overall computation time, little can be inferred from a specific example.

These observations, coupled with the fact that in most instances the algorithm quickly finds good points in the inducible region, suggest a variety of heuristics for speeding convergence. These include: 1) stopping after a given number of vertices, 2) stopping after a given number of times through Step 1 or Step 2, and 3) altering the branching rule. The first two follow from what appears to be a high correlation between the first and the last four columns in Table 4.1. With regard to the third, one branching rule that was tried involved ordering the variables according to their coefficient value in the leader's objective function. On average, however, a small degradation was experienced. Likewise, branching on the variables one at a time increased the computational effort anywhere from 50 to 1000 percent. Other possibilities are currently being investigated.

The use of alternative 0-1 programming algorithms begs investigation. The use of more advanced algorithms such as ZOOM which is the 0-1 version of the linear programming package XMP (Marsten 1981) may improve the

performance of the 0-1 BLPP algorithm. There also may be better ways to approach the solution of the 0-1 BLPP. Alternative formulations may prove more dynamic than the right-hand parametric formulation developed in Chapter 4.

Applications of 0-1 BLPP should also be explored. One area where research may prove fruitful is in developing road systems. The leader could be a builder wishing to keep road construction costs low while the follower could be the driving public who wish to minimize travel time. Many other uses for the 0-1 BLPP are awaiting the interested researcher.

8.3 The Continuous Variable BLPP

Experience has shown that even for the simplest of formulations, the bilevel programming problem is inherently difficult to solve. The branch and bound algorithm developed in this chapter attempts to exploit some of the special structure in the problem, and in so doing, is able to achieve rapid convergence. Linear problems with up to 100 variables and 40 constraints can be solved in less than 300 seconds on average. After thorough testing, the algorithm's performance and

robustness is seen to be superior to virtually all contenders.

In fact, one of the main advantages of the branch and bound approach is that it is quite general. The analysis centered on the linear formulation. Solutions to the nonlinear problem still require study. A possible approach is substituting a nonlinear computer code such as MINOS or GRG2 for the XMP library. Another area requiring exploration is the use of XMP in the branch and bound scheme. Although the capabilities of XMP are utilized, the development of other representations of the basis inverse and the data structure may improve the efficiency of the BLPP algorithm.

8.4 The Mixed Integer BLPP

This area requires further research. Although the algorithm does not guarantee the uncovering of the optimal solution, a more rigorous study of when and why the algorithm does not find the optimal solution is needed. Currently, if the algorithm cannot find the optimal solution of the mixed integer BLPP, uncertainty remains as to whether or not the problem has an optimal

solution. The development of a test to determine if a problem has an optimal solution is needed.

Computational experience shows the algorithm is limited in its ability to find feasible solutions of the mixed integer BLPP. The sizes of the problems solved are modest, and the CPU times required to solve these problems were sometimes large. An underlying reason for the large CPU times is that the algorithm must solve so many problems. At each vertex of the branch and bound tree, a BLPP is solved. If this solution satisfies the integer requirements, the follower's problem must also be solved. When the integer requirements are satisfied, the algorithm cannot backtrack; it continues to branch.

Improvements in computational efficiency are needed. If the algorithm for the continuous variable BLPP can be improved, the mixed integer BLPP algorithm would improve. Other branching schemes need to be explored. The current algorithm uses a depth only scheme. A breadth only approach may prove useful, and if a fathoming rule can be derived for this approach, the capabilities of the algorithm would be enhanced. A test which might prove useful is one which would determine when further branching cannot yield an

improvement in the leader's objective function. Also, greater use of the coefficients of the two objective functions might lead to improved branching rules.

8.5 Applications

The application in Chapter 7 highlights an area where use of the BLPP may be appropriate. It also demonstrates the care which must be used when trying to formulate a model using the BLPP. Although control of inventory and shortages would appear to belong to the manufacturer, they must be controlled by the customer because of the rules which must be observed in applying the BLPP. The application also emphasizes the need for improvement in the computational efficiency of the mixed integer BLPP algorithm.

8.6 Lessons Learned

The development of algorithms which solve various forms of the BLPP has been a trying experience. The BLPP is conceptually easy to grasp, but its modeling and application are difficult. When success appeared near, chasms appeared to block the way. The mixed integer BLPP had many of these. The discovery of simple problems which have no optimal solution, the inability to fathom subproblems with integer solutions, and the

need to accommodate the fact that the leader does not always take advantage of the integrality of the follower's variables hindered the development of an efficient algorithm. Also, determining the best way to order the algorithm for the 0-1 BLPP was a frustrating task.

Some wisdom was gained in this experience. Just because an algorithm and its associated computer code appears to solve a problem is no reason to accept it as flawless. Extensive testing and application is required. Even this may only raise the confidence that a flawed algorithm works. The discovery of problems which the mixed integer BLPP algorithm did not solve occurred only after many problems had been solved. Future work with the BLPP and other areas of operations research will be done with a critical eye and a reluctance to accept things as gospel. As the research developed, the possibilities seemed to expand. There are many areas which require more indepth study. As the effort progressed, the desire to return to the beginning and start over again sometimes became great. As the work invested in this study increased, it seemed to generate a need for more thought and work. Hopefully,

this work will prove to be useful and lead to further development of the body of work concerning the BLPP and the associated multilevel programming problem.

APPENDIX A

FORTRAN COMPUTER PROGRAMS

A.1 Introduction

In this appendix, the computer programs which are based on the algorithms developed in Chapters 4, 5 and 6 are presented. Listings of the programs used to generate the random problems and read data for example problems are also included in this appendix.

A.2 The Program Used to Generate Random Numbers

The following program was written by the computer consultants at the University of Texas at Austin and is designed to generate random numbers on an IBM computer. The program is based on the work of Knuth (1969). This program is inserted as a function in the computer programs which generate the random problems used to test the BLPP algorithms developed in Chapters 4, 5, and 6. The function is named URAND and is listed below.

REAL FUNCTION URAND (IY)
 INTEGER IY

C
 C URAND IS A UNIFORM RANDOM NUMBER GENERATOR BASED
 C ON THEORY AND SUGGESTIONS GIVEN IN D.E.
 C KNUTH (1969), VOL 2. THE INTEGER IY SHOULD
 C BE INITIALIZED TO AN ARBITRARY INTEGER PRIOR TO
 C THE FIRST CALL TO URAND. THE CALLING PROGRAM
 C SHOULD NOT ALTER THE VALUE OF IY BETWEEN
 C SUBSEQUENT CALLS TO URAND. VALUES OF URAND WILL
 C BE RETURNED IN THE INTERVAL (0,1).
 C

INTEGER IA,IC,ITWO,M2,M,MIC
 DOUBLE PRECISION HALFM
 REAL S
 DOUBLE PRECISION DATAN,DSQRT
 DATA M2/0/,ITWO/2/
 IF (M2.NE.0) GO TO 20

C
 C IF FIRST ENTRY, COMPUTE MACHINE INTEGER WORD LENGTH
 C

M=1
 10 M2=M
 M=ITWO*M2
 IF (M.GT.M2) GO TO 10
 HALFM=M2

C
 C COMPUTE MULTIPLIER AND INCREMENT FOR LINEAR
 C CONGRUENTIAL METHOD
 C

IA=8*IDINT(HALFM*DATAN(1.DO)/8.DO)+5
 IC=2*IDINT(HALFM*(0.5DO-DSQRT(3.DO)/6.DO))+1
 MIC=(M2-IC)+M2

C
 C S IS THE SCALE FACTOR FOR CONVERTING TO
 C FLOATING POINT
 C

S=0.5/HALFM

C
 C COMPUTE NEXT RANDOM NUMBER
 C

20 IY=IY*IA

C
 C THE FOLLOWING STATEMENT IS FOR COMPUTERS WHICH
 C DO NOT ALLOW INTEGER OVERFLOW ON ADDITION

```
C      IF (IY.GT.MIC) IY=(IY-M2)-M2
C
C      IY=IY+IC
C
C      THE FOLLOWING STATEMENT IS FOR COMPUTERS WHERE THE
C      WORD LENGTH FOR ADDITION IS GREATER THAN FOR
C      MULTIPLICATION
C
C      IF (IY/2.GT.M2) IY=(IY-M2)-M2
C
C      THE FOLLOWING STATEMENT IS FOR COMPUTERS WHERE
C      INTEGER OVERFLOW AFFECTS THE SIGN BIT
C
C      IF (IY.LT.0) IY=(IY+M2)+M2
C      URAND=FLOAT(IY)*S
C      RETURN
C      END
```

A.3 Computer Programs for the 0-1 BLPP

Three computer programs were developed to support the algorithm for the 0-1 BLPP. They are listed below. The first is the program which generated the random problems which were used to test the 0-1 BLPP algorithm. The second program is designed to read the data of a 0-1 BLPP and create parametric formulation used by the 0-1 BLPP algorithm. The third program listed below is the computerization of the 0-1 BLPP algorithm.

A.3.1 Program for Generating Random 0-1 BLPP

The first program is the one which generates the random problems used to test the 0-1 BLPP algorithm. The function URAND is called by this program and is presented in section A.2. This program requires five data entries. The entries are

- the number of variables controlled by the leader;
- the number of variables controlled by the follower;
- the number of constraints in the problem;
- the maximum number of nonzero entries in any one column of the A or B matrix;
- the random number seed.

The program does a free formatted read of the above input. Given this data, the program creates a 0-1 BLPP

in its parametric formulation. This problem can then be solved using the algorithm for the 0-1 BLPP. The variables of the program are documented in the listing which follows.

PROGRAM PURGEN

*****PURPOSE

PROGRAM GENERATES A RANDOM INTEGER BLPP WHERE THE VARIABLES ARE BINARY. THE FUNCTION URAND GENERATES THE RANDOM NUMBERS. SEE THE SECTION ON URAND FOR A LISTING OF THIS CODE.

VARIABLE DEFINITIONS

ACC ARRAY WHICH CONTAINS THE COEFFICIENTS OF EACH VARIABLE IN THE A-MATRIX OF THE PARAMETRIC FORMULATION OF THE 0-1 BLPP.

B THE RIGHT HAND SIDE ARRAY. B(1) IS THE RIGHT-HAND SIDE OF THE LEADER'S OBJECTIVE FUNCTION AND CORRESPONDS TO ALPHA. B(2) IS THE SUM OF THE VARIABLES CONTROLLED BY THE LEADER AND CORRESPONDS TO BETA.

CJ1 ARRAY CONTAINING THE LEADER'S OBJECTIVE FUNCTION COEFFICIENTS.

CJ2 ARRAY CONTAINING THE FOLLOWER'S OBJECTIVE FUNCTION COEFFICIENTS.

COLLEN THE NUMBER OF NONZEROS IN EACH COLUMN OF THE PARAMETRIC FORMULATION OF THE 0-1 BLPP.

IY RANDOM NUMBER SEED OF IY.

M NUMBER OF CONSTRAINTS IN THE PARAMETRIC FORMULATION OF THE 0-1 BLPP.

MAXCOL MAXIMUM NUMBER OF NONZEROS IN ANY COLUMN OF THE 0-1 BLPP.

MINMAX FLAG INDICATING NATURE OF THE PROBLEM. 1 MEANS MAXIMIZATION; -1 MEANS MINIMIZATION. FIXED AT 1 IN THIS PROGRAM.

NUMCON THE NUMBER OF CONSTRAINTS IN THE 0-1 BLPP.

NUMITR MAXIMUM NUMBER OF VERTICES ALLOWED IN THE BRANCH AND BOUND TREE.

NUMX THE NUMBER OF VARIABLES CONTROLLED BY THE LEADER.

NUMXY THE NUMBER OF VARIABLES IN THE 0-1 BLPP.

NUMY THE NUMBER OF VARIABLES CONTROLLED BY THE FOLLOWER.

```

C      ROWTYP  ARRAY WHICH INDICATES THE TYPE OF
C              CONSTRAINT:
C              +1 MEANS LESS THAN OR EQUAL TO;
C              0 MEANS EQUATION;
C              -1 MEANS GREATER THAN OR EQUAL TO.
C      ***END VARIABLE DEFINITIONS***
C
      INTEGER CNT,COLLEN(200),ACC(200,200),B(200),
1      CJ1(200),CJ2(200),ROWTYP(200),YA
      CHARACTER*4 DUMY
      DUMY='XSUB'
      READ (5,*) NUMX,NUMY,NUMCON,NUMITR,MAXCOL,IY
      MINMAX=1
      M=NUMCON+2
      B(1)=-9999
      B(2)=0
      ROWTYP(1)=-1
      ROWTYP(2)=-1
C      ***RANDOMLY GENERATE THE RIGHT HAND SIDE ARRAY
C      AND SET ROWTYPE EQUAL TO 1. ROWTYPE CAN BE
C      ALTERED BY MODIFYING THIS LOOP***
      DO 100 I=3,M
        ROWTYP(I)=1
        B(I)=URAND(IY)*45.0
100  CONTINUE
        NUMX1=NUMX+1
        NUMXY=NUMX+NUMY
C      ***RANDOMLY GENERATE THE COEFFICIENTS OF THE
C      LEADER'S AND FOLLOWER'S OBJECTIVE FUNCTIONS***
      DO 120 I=1,NUMXY
        IF (I.LE.NUMX) GO TO 110
        CJ2(I)=URAND(IY)*31.0-15.0
110    CJ1(I)=URAND(IY)*26.0-10.0
120  CONTINUE
        MAXC2=MAXCOL+MAXCOL+4
C
C      ***RANDOMLY GENERATE THE COEFFICIENTS OF THE
C      A- AND B-MATRICES***
C
      DO 130 I=1,NUMXY
        IROW=2
        CNT=0
        JSTART=0
        IF (CJ1(I).EQ.0) GO TO 131
        JSTART=JSTART+2

```

```

ACC(I,JSTART-1)=1
ACC(I,JSTART)=CJ1(I)
CNT=CNT+1
131 IF (I.GT.NUMX) GO TO 139
138 JSTART=JSTART+2
ACC(I,JSTART-1)=2
ACC(I,JSTART)=1
CNT=CNT+1
139 JSTART=JSTART+1
DO 140 J=JSTART,MAXC2,2
141 YA=URAND(IY)*29.0-10.0
IF (YA.EQ.0) GO TO 141
C ***ADJUST THE URAND(IY) MULTIPLIER IN THE FOLLOWING
C LINE TO CHANGE THE PROBLEM DENSITY. CHANGING
C MAXCOL ALSO EFFECTS THE PROBLEM DENSITY***
ROWNUM=URAND(IY)*3.0+1.0
IROW=ROWNUM+IROW
IF(IROW.GT.M) GO TO 145
ACC(I,J)=IROW
ACC(I,J+1)=YA
CNT=CNT+1
140 CONTINUE
145 COLLEN(I)=CNT
130 CONTINUE
C
C ***THE REMAINDER OF THE PROGRAM WRITES THE
C RANDOMLY GENERATED DATA TO A FILE. THE DATA
C IS FORMATTED SO IT CAN BE READ IN BY THE
C ALGORITHM USED TO SOLVE THE 0-1 BLPP***
C
WRITE (6,908) NUMX,NUMY,M,NUMITR
908 FORMAT(4I5)
DO 150 I=1,M
WRITE (6,910) I,ROWTYP(I),B(I)
150 CONTINUE
910 FORMAT(3I5)
DO 170 I=1,NUMXY
ICOLL=COLLEN(I)
ICOLL2=2*ICOLL
WRITE (6,915) DUMY,I,ICOLL,CJ2(I)
DO 180 J=1,ICOLL2,2
IB=ACC(I,J)
WRITE (6,920) IB,ACC(I,J+1)
180 CONTINUE
170 CONTINUE

```



```
915 FORMAT(A4,3I5)
920 FORMAT(2I5)
    STOP
    END
```

A.3.2 Program Which Creates the Parametric Formulation
of a 0-1 BLPP

The following program is used to read the data of a 0-1 BLPP and create its parametric formulation. The program's variables are defined in the code, and instructions for using the program are also included in the listing.

PROGRAM READ01

*****PURPOSE

THIS PROGRAM READS IN A 0-1 BILEVEL PROGRAMMING PROBLEM (BLPP) AND CREATES THE MATHEMATICAL PROGRAMMING PROBLEM WHICH CAN BE SOLVED TO FIND THE SOLUTION OF THE 0-1 BLPP.

*****DIRECTIONS FOR CREATING AN INPUT FILE FOR THIS PROGRAM

TO CREATE A REPRESENTATION OF THE BLPP WHICH CAN BE SOLVED USING THE ALGORITHM PRESENTED IN CHAPTER 4, ENTER THE DATA OF THE 0-1 BLPP AS OUTLINED BELOW.

THE PROBLEM SHOULD BE A MAXIMIZATION PROBLEM. THE VARIABLES SHOULD BE IN A SPECIFIC ORDER. THE VARIABLES CONTROLLED BY THE LEADER SHOULD PRECEDE THOSE CONTROLLED BY THE FOLLOWER.

ALL DATA IS INPUT USING FREE FORMAT. THIS MEANS DATA ON A CARD ARE SEPARATED BY COMMAS OR SPACES. EACH "CARD" OR LINE OF DATA REPRESENTS AN INPUT FIELD.

*
* NOTE: ALL ENTRIES SHOULD BE INTEGER. *
*

THE FIRST LINE OF DATA HAS FOUR ENTRIES. THESE ENTRIES PROVIDE INFORMATION ABOUT PROBLEM STRUCTURE.

THE THREE ENTRIES ARE:

ENTRY 1: NUMBER OF VARIABLES CONTROLLED BY THE LEADER;
ENTRY 2: NUMBER OF VARIABLES CONTROLLED BY THE FOLLOWER;
ENTRY 3: NUMBER OF CONSTRAINTS IN THE PROBLEM;
ENTRY 4: MAXIMUM NUMBER OF VERTICES ALLOWED IN THE BRANCH AND BOUND TREE.

C THE NEXT SECTION OF ENTRIES PROVIDES INFORMATION
 C ON THE CONSTRAINTS. THE INFORMATION IS
 C CONSTRAINT NUMBER, CONSTRAINT TYPE, AND THE
 C RIGHT-HAND-SIDE. THE CONSTRAINT TYPES ARE AS
 C FOLLOWS:

C +1--LESS THAN OR EQUAL TO;
 C 0--EQUALITY CONSTRAINT;
 C -1--GREATER THAN OR EQUAL TO;

C THE RIGHT-HAND SIDE ENTRY MUST BE POSITIVE.
 C THERE IS ONE LINE OF DATA PER CONSTRAINT.
 C ENTRY 1: ENTER THE CONSTRAINT NUMBER;
 C ENTRY 2: ENTER THE CONSTRAINT TYPE
 C (EITHER -1, 0, OR +1);
 C ENTRY 3: ENTER THE RIGHT-HAND-SIDE
 C OF THE CONSTRAINT (≥ 0).

C ENTER INFORMATION ON EACH VARIABLE. THE INPUTS
 C ARE THE VARIABLE'S POSITION IN THE INPUT SEQUENCE,
 C THE NUMBER OF NONZERO COEFFICIENTS IN THE
 C VARIABLE'S COLUMN, THE VARIABLE'S COEFFICIENT IN
 C THE LEADER'S OBJECTIVE FUNCTION, AND THE
 C VARIABLE'S COEFFICIENT IN THE FOLLOWER'S
 C OBJECTIVE FUNCTION. AFTER THIS LINE OF
 C INFORMATION FOR A VARIABLE IS ENTERED, THEN
 C ENTER ONE LINE OF DATA FOR EACH NONZERO
 C COEFFICIENT IN THE VARIABLE'S COLUMN. TWO VALUES
 C ARE REQUIRED: THE ROW NUMBER AND THE COEFFICIENT.
 C AFTER THIS INFORMATION HAS BEEN ENTERED FOR A
 C VARIABLE, PROCEED TO THE NEXT VARIABLE AND ENTER
 C THE INFORMATION ON IT.

C THE FIRST FOUR ENTRIES ARE:
 C ENTRY 1: THE VARIABLE NUMBER;
 C ENTRY 2: NUMBER OF NON-ZERO COEFFICIENTS IN THE
 C VARIABLE'S COLUMN IN THE A OR B MATRIX;
 C ENTRY 3: THE VARIABLE'S COEFFICIENT IN THE
 C LEADER'S OBJECTIVE FUNCTION;
 C ENTRY 4: THE VARIABLE'S COEFFICIENT IN THE
 C FOLLOWER'S OBJECTIVE FUNCTION (NOTE:
 C IF THE VARIABLE IS CONTROLLED BY THE
 C LEADER, AN ENTRY OF 0 IS PERMITTED).

C ENTER THE VARIABLE'S NON-ZERO COEFFICIENTS IN THE
 C A OR B MATRIX. TO ENTER THIS INFORMATION, TWO
 C DATA ENTRIES PER LINE ARE MADE.
 C
 C ENTRY 1: THE ROW NUMBER IN WHICH THE COEFFICIENT
 C APPEARS;
 C ENTRY 2: THE COEFFICIENT.
 C
 C AFTER ALL NON-ZERO COEFFICIENTS OF A VARIABLE HAVE
 C BEEN ENTERED, GO TO THE NEXT VARIABLE AND ENTER
 C INFORMATION FOR IT.
 C *****END OF DIRECTIONS*****
 C
 C *****VARIABLES DEFINITIONS*****
 C
 C ACC ARRAY WHICH CONTAINS THE COEFFICIENTS OF
 C EACH VARIABLE IN THE A- OR B-MATRIX OF THE
 C PARAMETRIC FORMULATION OF THE 0-1 BLPP.
 C B THE RIGHT HAND SIDE ARRAY. B(1) IS
 C ASSOCIATED WITH THE LEADER'S OBJECTIVE
 C FUNCTION AND CORRESPONDS TO ALPHA. B(2)
 C IS ASSOCIATED WITH THE SUM OF THE
 C VARIABLES CONTROLLED BY THE LEADER AND
 C CORRESPONDS TO BETA.
 C CJ1 ARRAY CONTAINING THE LEADER'S OBJECTIVE
 C FUNCTION COEFFICIENTS.
 C CJ2 ARRAY CONTAINING THE FOLLOWER'S OBJECTIVE
 C FUNCTION COEFFICIENTS.
 C COLLEN THE NUMBER OF NONZEROS IN EACH COLUMN OF
 C THE PARAMETRIC FORMULATION OF THE 0-1
 C BLPP.
 C CONX NUMBER OF CONSTRAINTS CONTAINING NO
 C VARIABLES CONTROLLED BY THE FOLLOWER.
 C CONY NUMBER OF CONSTRAINTS CONTAINING VARIABLES
 C CONTROLLED BY THE FOLLOWER.
 C M NUMBER OF CONSTRAINTS IN THE PARAMETRIC
 C FORMULATION OF THE 0-1 BLPP.
 C MINMAX FLAG INDICATING NATURE OF THE PROBLEM.
 C 1 MEANS MAXIMIZATION AND -1 MEANS
 C MINIMIZATION. FIXED AT 1 IN THIS PROGRAM.
 C NUMCON THE NUMBER OF CONSTRAINTS IN THE 0-1 BLPP.
 C NUMITR THE MAXIMUM NUMBER OF VERTICES IN THE
 C BRANCH AND BOUND TREE CREATED WHEN SOLVING
 C THE 0-1 BLPP.

```

C      NUMX      THE NUMBER OF VARIABLES CONTROLLED BY THE
C      LEADER.
C      NUMXY     THE NUMBER OF VARIABLES IN THE 0-1 BLPP.
C      NUMY      THE NUMBER OF VARIABLES CONTROLLED BY THE
C      FOLLOWER.
C      ROWTYP    ARRAY WHICH INDICATES THE CONSTRAINT TYPE
C      +1 MEANS LESS THAN OR EQUAL TO;
C      0 MEANS EQUATION;
C      -1 MEANS GREATER THAN OR EQUAL TO.
C      *****END VARIABLE DEFINITIONS*****
C
      INTEGER COLLEN(200),ACC(200,200),B(200),
1      CJ1(200),CJ2(200),ROWTYP(200)
      CHARACTER*4 DUMY
      DUMY='XSUB'
C      ***READ PROBLEM INFORMATION***
      READ (5,*) NUMX,NUMY,NUMCON,NUMITR
      MINMAX=1
      M=NUMCON+2
      NUMX1=NUMX+1
      NUMXY=NUMX+NUMY
C      ***SET B(1) AND B(2)--THEY CORRESPOND TO ALPHA AND
C      BETA IN THE ALGORITHM.***
      B(1)=-9999
      B(2)=0
      ROWTYP(1)=-1
      ROWTYP(2)=-1
C      ***READ IN DATA ON THE CONSTRAINTS***
      DO 120 I=3,M
          READ (5,*) NUMCON,ROWTYP(I),B(I)
120  CONTINUE
C      ***READ IN DATA ON THE VARIABLES***
      K=1
130  READ (5,*) NUMVAR,COLLEN(K),CJ1(K),CJ2(K)
      LSTART=0
      IF (CJ1(K).EQ.0) GO TO 50
      LSTART=LSTART+2
      COLLEN(K)=COLLEN(K)+1
      ACC(K,LSTART-1)=1
      ACC(K,LSTART)=CJ1(K)
50  CONTINUE
      IF (K.GT.NUMX) GO TO 60
      LSTART=LSTART+2
      ACC(K,LSTART-1)=2
      ACC(K,LSTART)=1

```

```

COLLEN(K)=COLLEN(K)+1
60 CONTINUE
COLL2=COLLEN(K)+COLLEN(K)
LSTART=LSTART+1
DO 140 L=LSTART,COLL2,2
  READ (5,*) ACC(K,L),ACC(K,L+1)
  ACC(K,L)=ACC(K,L)+2
140 CONTINUE
  K=K+1
  IF (K.GT.NUMXY) GO TO 145
  GO TO 130
145 CONTINUE
C
C   ***END DATA INPUT***
C
C   ***BEGIN CREATION OF THE DATA SET FOR 0-1 BLPP
C   ALGORITHM***
C
C   ***WRITE BASIC PROBLEM INFORMATION***
  WRITE (6,908) NUMX,NUMY,M,NUMITR
908 FORMAT(4I5)
C   ***WRITE CONSTRAINT INFORMATION***
  DO 150 I=1,M
    WRITE (6,910) I,ROWTYP(I),B(I)
150 CONTINUE
910 FORMAT(3I5)
C   ***WRITE VARIABLE INFORMATION***
  DO 170 I=1,NUMXY
    ICOLL=COLLEN(I)
    ICOLL2=2*ICOLL
    WRITE (6,915) DUMY,I,ICOLL,CJ2(I)
    DO 180 J=1,ICOLL2,2
      IB=ACC(I,J)
      WRITE (6,920) IB,ACC(I,J+1)
180 CONTINUE
170 CONTINUE
915 FORMAT(A4,3I5)
920 FORMAT(2I5)
  STOP
  END

```

A.3.3 The Algorithm for the 0-1 BLPP

The following program is the computerized version of the algorithm for the 0-1 BLPP. The input for the program is a 0-1 BLPP which is formulated in its parametric right-hand side representation. The program of section A.3.1 creates random 0-1 BLPPs in this form, and the program of section A.3.2 converts the data of a 0-1 BLPP into the proper form. The program which solves the 0-1 BLPP follows and is documented in line.

PROGRAM BLPP01

*****PURPOSE

THE PROGRAM READS IN THE PARAMETRIC FORMULATION OF THE 0-1 BLPP AND FINDS THE SOLUTION OF THE 0-1 BLPP PROBLEM. THE DATA IS REFORMATTED SO A 0-1 INTEGER PROGRAMMING PROBLEM USING A BALAS ADDITIVE ALGORITHM CAN BE USED.

VARIABLE DEFINITIONS

A ARRAY CONTAINING THE NON-ZERO COEFFICIENTS OF THE VARIABLES IN THE A- AND B-MATRICES.

B THE RIGHT HAND SIDE ARRAY. B(1) IS ASSOCIATED WITH THE LEADER'S OBJECTIVE FUNCTION AND CORRESPONDS TO ALPHA. B(2) IS ASSOCIATED WITH THE SUM OF THE VARIABLES CONTROLLED BY THE LEADER AND CORRESPONDS TO BETA.

BIFLG FLAG USED BY 0-1 SOLVER. 0 MEANS FIND THE OPTIMAL SOLUTION. 1 MEANS FIND A FEASIBLE SOLUTION.

BSTNOD VERTEX AT WHICH INCUMBENT SOLUTION FOUND.

CJ1 ARRAY CONTAINING THE LEADER'S OBJECTIVE FUNCTION COEFFICIENTS.

CJ2 ARRAY CONTAINING THE FOLLOWER'S OBJECTIVE FUNCTION COEFFICIENTS. THESE COEFFICIENTS ARE THE COEFFICIENTS OF THE OBJECTIVE FUNCTION OF THE PARAMETRIC FORMULATION OF THE 0-1 BLPP.

COL ARRAY CONTAINING THE STARTING POSITION IN A OF EACH VARIABLE'S COLUMN.

COLLEN THE NUMBER OF NONZEROS IN EACH COLUMN OF THE PARAMETRIC FORMULATION OF THE 0-1 BLPP.

DRHS ARRAY USED IN BALAS ADDITIVE ALGORITHM. WORKING ARRAY VERSION OF RHS USED TO CHECK SOLUTION FEASIBILITY AND CHOOSE A SEPARATION VARIABLE.

FDUM USED IN BALAS ADDITIVE ALGORITHM. SUMS THE ELEMENTS OF ARRAY RHS WHICH ARE NEGATIVE. USED TO CHECK FOR FEASIBILITY OF CURRENT VALUES OF VARIABLES.

FEAS USED IN BALAS ADDITIVE ALGORITHM. A FLAG USED TO INDICATE IF SEPARATING ON A FREE

C VARIABLE IS FEASIBLE. 0 MEANS STILL
 C FEASIBLE. NOT 0 MEANS INFEASIBLE.
 C IDUM USED IN BALAS ADDITIVE ALGORITHM TO
 C DETERMINE THE CONTRIBUTION OF EACH FREE
 C VARIABLE TO PROBLEM INFEASIBILITY.
 C IEQ ARRAY USED TO CONVERT THE PARAMETRIC
 C FORMULATION OF THE 0-1 BLPP INTO A FORM
 C COMPATIBLE WITH BALAS' ADDITIVE ALGORITHM.
 C USED TO DEAL WITH EQUALITY CONSTRAINTS.
 C INF USED IN BALAS ADDITIVE ALGORITHM. IT IS
 C THE CONTRIBUTION TO INFEASIBILITY OF EACH
 C FREE VARIABLE.
 C J USED IN THE BALAS ADDITIVE ALGORITHM. IT
 C IS THE INDEX OF THE SEPARATION VARIABLE.
 C KL CURRENT TREE LEVEL IN THE INNER TREE.
 C M NUMBER OF CONSTRAINTS IN THE PARAMETRIC
 C FORMULATION OF THE 0-1 BLPP.
 C NODE NUMBER OF VERTICES IN OUTER BRANCH AND
 C BOUND TREE.
 C NUMCON THE NUMBER OF CONSTRAINTS IN THE 0-1 BLPP.
 C NUMX THE NUMBER OF VARIABLES CONTROLLED BY THE
 C LEADER.
 C NUMXY THE NUMBER OF VARIABLES IN THE 0-1 BLPP.
 C NUMY THE NUMBER OF VARIABLES CONTROLLED BY THE
 C FOLLOWER.
 C OUTLEV LEVEL OF THE CURRENT VERTEX IN THE OUTER
 C BRANCH AND BOUND TREE.
 C RHS ARRAY USED IN BALAS ADDITIVE ALGORITHM
 C TO STORE THE CURRENT RIGHT-HAND SIDE OF
 C EACH CONSTRAINT.
 C ROWNOS ARRAY CONTAINING THE ROW NUMBER OF EACH
 C NONZERO COEFFICIENT IN THE A- AND B-
 C MATRICES.
 C ROWTYP ARRAY WHICH INDICATES THE TYPE OF
 C CONSTRAINT:
 C +1 MEANS LESS THAN OR EQUAL TO;
 C 0 MEANS EQUATION;
 C -1 MEANS GREATER THAN OR EQUAL TO.
 C S ARRAY USED IN BALAS ADDITIVE ALGORITHM
 C SHOWING THE SEPARATION VARIABLES OF THE
 C INNER BRANCH AND BOUND TREE. S(K) IS THE
 C INDEX OF SEPARATION VARIABLE AT LEVEL K.
 C SO ARRAY SHOWING THE SEPARATION VARIABLES OF
 C OUTER BRANCH AND BOUND TREE. SO(I) IS THE
 C INDEX OF SEPARATION VARIABLE AT LEVEL I.

C T ARRAY USED IN BALAS ADDITIVE ALGORITHM
 C WHICH CONTAINS THE SUM OF THE NEGATIVE
 C COEFFICIENTS OF THE FREE VARIABLES IN
 C EACH CONSTRAINT.
 C TRANS ARRAY USED TO TRANSFORM SOLUTION PROVIDED
 C BY BALAS' ADDITIVE ALGORITHM.
 C V ARRAY USED IN BALAS ADDITIVE ALGORITHM
 C INDICATING WHETHER OR NOT THE
 C ALTERNATIVE VERTICES HAVE BEEN EXPLORED.
 C V(I)=0 MEANS THE ALTERNATIVE HAS NOT BEEN
 C EXPLORED AT LEVEL I;
 C V(I)=1 MEANS THE ALTERNATIVE HAS BEEN
 C EXPLORED.
 C VO ARRAY INDICATING WHETHER OR NOT THE
 C ALTERNATIVE VERTICES HAVE BEEN EXPLORED.
 C VO(I)=0 MEANS THE ALTERNATIVE HAS NOT BEEN
 C EXPLORED AT LEVEL I;
 C VO(I)=1 MEANS THE ALTERNATIVE HAS BEEN
 C EXPLORED.
 C W ARRAY USED IN BALAS ADDITIVE ALGORITHM.
 C W(I)=0 MEANS VARIABLE I IS FREE;
 C W(I)=-K MEANS VARIABLE I SET TO 0 AT LEVEL
 C K OF INNER BRANCH AND BOUND TREE;
 C W(I)=K MEANS VARIABLE I SET TO 1 AT LEVEL
 C K OF INNER BRANCH AND BOUND TREE.
 C XBEST ARRAY WHICH STORES THE INCUMBENT SOLUTION
 C OF THE 0-1 BLPP.
 C XFEAS FLAG WHICH INDICATES FEASIBILITY OF THE
 C 0-1 PROBLEM AT THE CURRENT VERTEX OF THE
 C OUTER BRANCH AND BOUND TREE. 0 MEANS
 C FEASIBLE. 1 MEANS INFEASIBLE.
 C XFIX ARRAY USED TO FIX THE VARIABLES CONTROLLED
 C BY THE LEADER AT 0 OR 1.
 C XSET ARRAY WHICH TRACKS WHICH VARIABLES
 C CONTROLLED BY THE LEADER ARE SET TO 0 OR
 C 1 IN THE OUTER BRANCH AND BOUND TREE.
 C XSET(J)=0 MEANS VARIABLE J IS FREE,
 C XSET(J)<0 MEANS VARIABLE J SET TO 0,
 C XSET(J)>0 MEANS VARIABLE J SET TO 1.
 C XYBEST ARRAY WHICH STORES THE SOLUTION AT THE
 C CURRENT VERTEX OF THE OUTER BRANCH AND
 C BOUND TREE WHEN THE FREE VARIABLES
 C CONTROLLED BY THE LEADER ARE NOT FIXED.
 C Y ARRAY USED IN BALAS ADDITIVE ALGORITHM
 C TO STORE THE VALUES OF THE VARIABLES.

```

C      YBEST  ARRAY WHICH STORES THE SOLUTION AT THE
C            CURRENT VERTEX OF THE OUTER BRANCH AND
C            BOUND TREE WHEN THE FREE VARIABLES
C            CONTROLLED BY THE LEADER ARE FIXED
C            AT THEIR VALUE IN XYBEST.  THUS YBEST
C            ALWAYS CONTAINS A FEASIBLE SOLUTION OF
C            0-1 BLPP.  DETERMINED IN BALAS ADDITIVE
C            ALGORITHM.
C      YFEAS  ARRAY USED IN BALAS ADDITIVE ALGORITHM TO
C            STORE A FEASIBLE SOLUTION.
C      YJ     USED IN BALAS ADDITIVE ALGORITHM.  IT IS
C            THE DIRECTION OF EXPLORATION FOR THE
C            SEPARATION VARIABLE (0 OR 1).
C      Z1     VALUE OF THE INCUMBENT SOLUTION.
C      Z1A    VALUE OF THE SOLUTION AT THE CURRENT
C            VERTEX.
C      Z2BEST VALUE OF THE OPTIMAL SOLUTION FOUND BY THE
C            BALAS ADDITIVE ALGORITHM.
C      Z2     USED IN BALAS ADDITIVE ALGORITHM TO TEST
C            IF A SOLUTION CAN BE FATHOMED.
C      Z2FEAS USED IN BALAS ADDITIVE ALGORITHM TO TEST
C            FEASIBILITY OF CURRENT SOLUTION.
C      Z2FIX1 USED IN BALAS ADDITIVE ALGORITHM.  IT IS
C            THE SUM OF THE OBJECTIVE FUNCTION
C            COEFFICIENTS OF THOSE VARIABLES WHICH ARE
C            FIXED AT 1.
C      ***END VARIABLE DEFINITIONS***
C
C      *****NOTE*****
C      X REFERS TO VARIABLES CONTROLLED BY THE LEADER;
C      Y REFERS TO VARIABLES CONTROLLED BY THE FOLLOWER;
C      *****END OF NOTE*****
C
C      IMPLICIT INTEGER (A-Z)
C      REAL DEN,BIDEN
C      DIMENSION XFIX(100),XSET(200),VO(200),SO(200)
C      DIMENSION COL(200),ROWNOS(800),B(100),A(800)
C      DIMENSION ROWTYP(100),IEQ(100),TRANS(200)
C      DIMENSION W(200),V(200),S(200),RHS(100),DRHS(100)
C      DIMENSION YBEST(200),XYBEST(200),CJ2(200)
C      DIMENSION YFEAS(200),Y(200),T(100),CJ1(200)
C      DIMENSION XBEST(200)
C      CHARACTER*4 DUMY,XSUB

```

C

```

C      BIG=9999
      XSUB='XSUB'
C      ***READ IN PATION***
      READ (5,900) NUMX,NUMY,M,NUMITR
      WRITE(6,901) NUMX,NUMY,M
900  FORMAT(3I5)
901  FORMAT(' X VAR= ',I5,' Y VAR= ',I5,' CON= ',I5)
      NUMXY=NUMX+NUMY
      ASIZE=NUMXY*(M-2)
      MT=M
      DO 10 I=1,NUMXY
        TRANS(I)=0
C      10 CONTINUE
C
C      ***READ IN ROW TYPES AND RIGHT-HAND-SIDES
C      ISEQ REPRESENTS THE ROW NUMBER***
C
      IC=0
      DO 100 I=1,MT
        READ(5,902) ISEQ,ROWTYP(I),B(I)
        IF (ROWTYP(I).NE.0) GO TO 99
        IC=IC+2
        M=M+1
        ROWTYP(I)=1
        ROWTYP(M)=0
        IEQ(IC-1)=I
        IEQ(IC)=M
        B(M)=-B(I)
99      IF (ROWTYP(I).EQ.-1) B(I)=-B(I)
100     CONTINUE
902  FORMAT(3I5)
C
C      ***READ IN COLUMNS ONE AT A TIME AND UPDATE
C      MATRICES***
C
      ACCOUNT=0
      BICNT=0
      BISIZE=NUMXY*M
      COL(1)=1
      DO 120 I=1,NUMXY
        ICNT=0
        READ(5,904) DUMY,JSEQ,COLLEN,CJ2(I)
        IF (DUMY.NE.XSUB) GO TO 999
        COL(I+1)=COL(I)+COLLEN

```

```

KSTART=COL(I)
KSTOP=COL(I+1)-1
IF (KSTART.GT.KSTOP) GO TO 999
DO 130 K=KSTART,KSTOP
  READ(5,906) ROWNOS(K),A(K)
  BICNT=BICNT+1
  IJ=ROWNOS(K)
  IF (IJ.EQ.1) CJ1(I)=A(K)
  IF (IJ.GE.3) ACOUNT=ACOUNT+1
  IF (ROWTYP(IJ).EQ.1) GO TO 130
  IF (ROWTYP(IJ).EQ.-1) GO TO 129
  ICNT=ICNT+1
  COL(I+1)=COL(I+1)+1
  DO 125 IC=1,MT,2
    IF (IEQ(IC).EQ.IJ) GO TO 127
125  CONTINUE
127  ROWNOS(KSTOP+ICNT)=IEQ(IC+1)
    A(KSTOP+ICNT)=-A(K)
    GO TO 130
129  A(K)=-A(K)
130  CONTINUE
120 CONTINUE
904 FORMAT(A4,3I5)
906 FORMAT(2I5)
C
C  ***MAKE ALL OBJECTIVE FUNCTION COEFFICIENTS
C  NONPOSITIVE***
C
DO 200 I=NUMX,NUMXY
  IF (CJ2(I).LE.0) GO TO 200
  KSTART=COL(I)
  KSTOP=COL(I+1)-1
  DO 150 K=KSTART,KSTOP
    IJ=ROWNOS(K)
    B(IJ)=B(IJ)-A(K)
    A(K)=-A(K)
150  CONTINUE
    CJ2(I)=-CJ2(I)
    TRANS(I)=1
200 CONTINUE
C
C  ***THE OUTER LOOP BEGINS***
C  ***INITIALIZE VARIABLES***
OUTLEV=0
BIFLG=1

```

```

XFEAS=1
DO 250 I=1,NUMX
    SO(I)=0
    VO(I)=0
    XBEST(I)=0
    XSET(I)=0
    XYBEST(I)=0
    XFIX(I)=0
250 CONTINUE
    B(1)=9999
    Z2BEST=-9999
    NODE=0
    Z1=9999
    CALL CPTIME(TOTCP1,VIRCP1)
C    ***CALL THE BALAS ADDITIVE ALGORITHM***
305 CALL BALAS(XFIX,XFEAS,BIFLG,XYBEST,Z2BEST,B,
1          A,CJ2,COL,ROWNOS,BIG,NUMXY,
2          M,NUMX,NUMY)
C    ***IF PROBLEM IS INFEASIBLE, GO TO 500 AND
C    BACKTRACK***
    IF (XFEAS.NE.0) GO TO 500
    DO 307 I=1,NUMXY
        YBEST(I)=XYBEST(I)
307 CONTINUE
    IF (OUTLEV.EQ.0) GO TO 310
C    ***FIND A FEASIBLE SOLUTION OF THE 0-1 BLPP***
    CALL BIFEAS(XFIX,Z2BEST,YBEST,B,A,CJ2,COL,
1          ROWNOS,BIG,NUMXY,M,NUMX,NUMY)
C    ***CHECK FOR IMPROVEMENT ON CURRENT INCUMBENT
C    SOLUTION***
310 Z1A=0
    DO 320 I=1,NUMXY
        K=COL(I)
        IF (YBEST(I).EQ.0) GO TO 330
        IF (ROWNOS(K).NE.1) GO TO 330
        Z1A=Z1A+A(K)
320 CONTINUE
    IF (Z1A.GE.Z1) GO TO 335
C    ***UPDATE THE INCUMBENT SOLUTION***
    Z1=Z1A
    BSTNOD=NODE
    DO 330 I=1,NUMXY
        XBEST(I)=YBEST(I)
330 CONTINUE
C    ***INCREASE SUM OF X VARIABLES***

```

```

335 SUMX=0
    DO 340 I=1,NUMX
        SUMX=SUMX+YBEST(I)
340 CONTINUE
    IF (OUTLEV.EQ.0) XFIRST=SUMX
    B(1)=Z1-1
    B(2)=-SUMX-1
C    ***SELECT A BRANCHING VARIABLE***
    DO 360 I=1,NUMX
        IF (YBEST(I).EQ.0) GO TO 360
        IF (XSET(I).NE.0) GO TO 360
        OUTLEV=OUTLEV+1
        XSET(I)=OUTLEV
        SO(OUTLEV)=I
        NODE=NODE+1
360 CONTINUE
    GO TO 600
C
C    ***BACKTRACK--DOES BACKTRACK IN OUTER TREE***
C
500 IF (OUTLEV.EQ.0) GO TO 750
    IF (VO(OUTLEV).EQ.0) GO TO 550
    XSET(SO(OUTLEV))=0
    VO(OUTLEV)=0
    OUTLEV=OUTLEV-1
    GO TO 500
550 VO(OUTLEV)=1
    XSET(SO(OUTLEV))=-XSET(SO(OUTLEV))
    NODE=NODE+1
    B(1)=Z1-1
    B(2)=0
C    ***SET XFIX FOR BALAS ADDITIVE ALGORITHM***
600 IF (NODE.GT.NUMITR) GO TO 745
    DO 610 I=1,NUMX
        XFIX(I)=XSET(I)
610 CONTINUE
    BIFLG=1
    XFEAS=1
    Z2BEST=-9999
    GO TO 305
C    ***TRANSFORM VARIABLES TO ORIGINAL FORM FOR
C    OPTIMAL SOLUTION***
745 WRITE (6,746)
746 FORMAT(' THE FOLLOWING SOLUTION IS NOT OPTIMAL. ',
1      ' NODE LIMIT EXCEEDED. ')

```



```

750 CONTINUE
    CALL CPTIME(TOTCP2,VIRCP2)
    TOTCP=TOTCP2-TOTCP1
    VIRCP=VIRCP2-VIRCP1
    Z1=0
    DO 790 I=1,NUMXY
        IF (TRANS(I).EQ.0) GO TO 780
        IF (XBEST(I).EQ.0) GO TO 755
        XBEST(I)=0
        GO TO 790
755     XBEST(I)=1
780     IF (XBEST(I).EQ.0) GO TO 790
        Z1=Z1+CJ1(I)
790 CONTINUE
C     ***PRINT OUT SOLUTION***
    WRITE(6,910) Z1
910  FORMAT('    THE BEST SOLUTION IS',I8)
    WRITE(6,911)
911  FORMAT('    THE VALUE OF THE VARIABLES AT ',
1      'OPTIMALITY IS')
    DO 800 I=1,NUMXY
        WRITE(6,912) I,XBEST(I)
800  CONTINUE
912  FORMAT('    VARIABLE ',I3,' HAS VALUE ',I2)
    GO TO 950
999  WRITE(6,913)
913  FORMAT(' THE DRAGON GOT US LORD OSCAR')
    WRITE(6,976) NODE,BSTNOD
976  FORMAT(' NODES IN CONTROLLING TREE ',I5,' BEST ',
1      'NODE ',I5)
    DEN=FLOAT(ACOUNT)/FLOAT(ASIZE)
    BIDEN=FLOAT(BICNT)/FLOAT(BISIZE)
    WRITE (6,980) DEN,BIDEN
980  FORMAT(' DENSITY OF A MATRIX = ',F7.4,' DENSITY ',
1      'OF PROB SOLVED BY ALGOR = ',F7.4)
1000 WRITE (6,957) TOTCP,VIRCP
957  FORMAT(1H0,'TOTAL CPTIME =',I6,3X,'VIRTUAL ',
1      'CPTIME =',I6)
    STOP
    END
    SUBROUTINE BIFEAS(XFIX,Z2BEST,YBEST,B,A,CJ2,COL,
1      ROWNOS,BIG,NUMXY,M,NUMX,NUMY)
C     *****PURPOSE
C     THIS SUBROUTINES FIXES EACH VARIABLE CONTROLLED
C     BY THE LEADER AND CALLS THE BALAS ADDITIVE

```

```

C      ALGORITHM.  IF THE ALGORITHM RETURNS A SOLUTION,
C      THIS SOLUTION IS A FEASIBLE SOLUTION OF THE 0-1
C      BLPP.
      IMPLICIT INTEGER (A-Z)
      DIMENSION XFIX(100),COL(200),ROWNOS(800),B(100)
      DIMENSION A(800),CJ2(200),BI(2),YBEST(200)
      DO 100 I=1,NUMX
        XFIX(I)=I
        IF (YBEST(I).EQ.0) XFIX(I)=-I
100  CONTINUE
      B(1)=9999
      B(2)=0
      BIFLG=0
      XFEAS=1
      Z2BEST=-9999
      CALL BALAS(XFIX,XFEAS,BIFLG,YBEST,Z2BEST,
1          B,A,CJ2,COL,ROWNOS,BIG,NUMXY,
2          M,NUMX,NUMY)
      RETURN
      END
      SUBROUTINE BALAS(XFIX,XFEAS,BIFLG,YBEST,Z2BEST,
1          B,A,C2,COL,ROWNOS,BIG,NUMXY,
2          M,NUMX,NUMY)

C      *****PURPOSE
C      THIS IS A FORTRAN VERSION OF THE BALAS ADDITIVE
C      ALGORITHM FOR PURE 0-1 INTEGER PROGRAMMING.  IT IS
C      BASED ON CODES DEVELOPED BY DR. PAUL A. JENSEN.
C      IT IS USED TO SOLVE THE 0-1 INTEGER PROGRAMMING
C      PROBLEM DERIVED FROM THE 0-1 BLPP.

      IMPLICIT INTEGER (A-Z)
      DIMENSION XFIX(100),COL(200)
      DIMENSION ROWNOS(800),B(100),A(800),C2(200)
      DIMENSION W(200),V(200),S(200),RHS(100),DRHS(100)
      DIMENSION YFEAS(200),Y(200),T(100),YBEST(200)
      DO 1 I=1,NUMXY
        YBEST(I)=0
1  CONTINUE
      KL=0
      CALL SETUP(FEAS,Z2FIX1,Y,W,S,V,COL,
1          ROWNOS,A,T,RHS,B,M,NUMXY)
C      ***FIX THOSE VARIABLES SET IN OUTER TREE***
      DO 3 I=1,NUMX
        IF (XFIX(I).EQ.0) GO TO 3

```

```

        KL=KL+1
        J=I
        V(KL)=1
        YJ=1
        IF (XFIX(I).LT.0) YJ=0
        CALL SET(Z2FIX1,KL,W,S,Y,T,RHS,COL,
1          ROWNOS,A,C2,FEAS,YJ,J)
        IF (FEAS.NE.0) GO TO 99
3 CONTINUE
C*****
C BEGIN FATHOM SECTION*
C*****
2 CALL FTEST(COL,ROWNOS,A,RHS,T,W,V,KL,YJ,FEAS,
1  NUMXY,Z2FIX1,S,Y,C2,J)
  IF (FEAS.NE.0) GO TO 50
5 CALL RELAX(Z2FIX1,Z2)
  IF (Z2.EQ.BIG) GO TO 50
6 IF (Z2.LE.Z2BEST) GO TO 50
7 CALL ROUND(Z2FEAS,Z2,RHS,YFEAS,Y,DRHS,
1  FDUM,BIG,M,NUMXY)
  IF (Z2FEAS.LE.Z2BEST) GO TO 11
8 Z2BEST=Z2FEAS
  XFEAS=0
  DO 10 I=1,NUMXY
    YBEST(I)=YFEAS(I)
10 CONTINUE
  IF (BIFLG.EQ.1) GO TO 99
  IF (Z2FEAS.EQ.Z2) GO TO 50
11 CALL CHOOSE(COL,ROWNOS,FDUM,DRHS,A,W,YJ,
1  BIG,NUMXY,J)
  IF (J.EQ.0) GO TO 50
  KL=KL+1
  V(KL)=0
  CALL SET(Z2FIX1,KL,W,S,Y,T,RHS,COL,
1  ROWNOS,A,C2,FEAS,YJ,J)
  IF (FEAS.NE.0) GO TO 50
12 GO TO 2
C*****
C BEGIN BACKTRACK SECTION*
C*****
50 IF (KL.EQ.0) GO TO 99
  IF (V(KL).NE.0) GO TO 80
  J=S(KL)
  IF (W(J).GT.0) GO TO 55
  YJ=1

```

```

        GO TO 60
55 YJ=0
60 CALL RESET(KL,Y,W,V,S,Z2FIX1,T,RHS,COL,
1         ROWNOS,A,C2,NUMXY)
        V(KL)=1
        CALL SET(Z2FIX1,KL,W,S,Y,T,RHS,COL,ROWNOS,
1         A,C2,FEAS,YJ,J)
        IF (FEAS.NE.0) GO TO 50
        GO TO 2
80 KL=KL-1
        GO TO 50
99 RETURN
C*****
C  END OF SUBROUTINE BALAS*
C*****
        END
C
C*****
C BEGIN SUBROUTINE SETUP*
C*****
        SUBROUTINE SETUP(FEAS,Z2FIX1,Y,W,S,V,COL,ROWNOS,
1         A,T,RHS,B,M,NUMXY)
C  THIS SUBROUTINE INITIALIZES OR 'SETUP' VARIABLES
C
        IMPLICIT INTEGER (A-Z)
        DIMENSION COL(200),ROWNOS(800),B(100),A(800)
        DIMENSION W(200),V(200),S(200),RHS(100)
        DIMENSION Y(200),T(100)
        Z2FIX1=0
        FEAS=0
        DO 100 I=1,M
            RHS(I)=B(I)
            T(I)=0
100 CONTINUE
        DO 110 L=1,NUMXY
            Y(L)=0
            W(L)=0
            S(L)=0
            V(L)=0
            KSTART=COL(L)
            KSTOP=COL(L+1)-1
            DO 120 K=KSTART,KSTOP
                I=ROWNOS(K)
                IF(A(K).LT.0) T(I)=T(I)+A(K)
120 CONTINUE

```

```

110 CONTINUE
    RETURN
    END
C*****
C BEGIN SUBROUTINE RELAX*
C*****
    SUBROUTINE RELAX(Z2FIX1,Z2)
    IMPLICIT INTEGER (A-Z)
    Z2=Z2FIX1
    RETURN
    END
C*****
C BEGIN SUBROUTINE ROUND*
C*****
    SUBROUTINE ROUND(Z2FEAS,Z2,RHS,YFEAS,Y,DRHS,
1      FDUM,BIG,M,NUMXY)
C    THIS SUBROUTINE CHECKS THE FEASIBILITY OF
C    A SOLUTION
C
    IMPLICIT INTEGER (A-Z)
    DIMENSION RHS(100),DRHS(100)
    DIMENSION YFEAS(200),Y(200)
    Z2FEAS=Z2
    FDUM=0
    DO 130 I=1,M
        DRHS(I)=RHS(I)
        IF(DRHS(I).LT.0) FDUM=FDUM-DRHS(I)
130 CONTINUE
    IF (FDUM.GT.0) GO TO 140
    DO 135 IJ=1,NUMXY
        YFEAS(IJ)=Y(IJ)
135 CONTINUE
    RETURN
140 Z2FEAS=-BIG
    RETURN
    END
C*****
C BEGIN SUBROUTINE CHOOSE*
C*****
    SUBROUTINE CHOOSE(COL,ROWNOS,FDUM,DRHS,A,W,YJ,
1      BIG,NUMXY,J)
C    THIS SUBROUTINE CHOOSES THE BEST FREE VARIABLE
C    AND SETS IT TO 1
C
    IMPLICIT INTEGER (A-Z)

```

```

        DIMENSION COL(200),ROWNOS(800),A(800)
        DIMENSION W(200),DRHS(100)
        DM=BIG
        J=0
        L=0
150    L=L+1
        IF (L.GT.NUMXY) GO TO 199
        IF (W(L).NE.0) GO TO 150
        KSTART=COL(L)
        KSTOP=COL(L+1)-1
        INF=FDUM
        DO 160 K=KSTART,KSTOP
            I=ROWNOS(K)
            IDUM=DRHS(I)-A(K)
            IF (DRHS(I).LT.0) INF=INF+DRHS(I)
            IF (IDUM.LT.0) INF=INF-IDUM
160    CONTINUE
        IF (DM.LT.INF) GO TO 150
        DM=INF
        J=L
        GO TO 150
199    IF (J.EQ.0) GO TO 200
        YJ=1
200    RETURN
        END
C*****
C BEGIN SUBROUTINE SET*
C*****
        SUBROUTINE SET(Z2FIX1,KL,W,S,Y,T,RHS,COL,
1          ROWNOS,A,C2,FEAS,YJ,J)
C    SUBROUTINE SET SETS VARIABLES TO 0 OR 1 IF
C    BACK TRACKING
C
        IMPLICIT INTEGER (A-Z)
        DIMENSION COL(200),ROWNOS(800),A(800),C2(200)
        DIMENSION W(200),S(200),RHS(100)
        DIMENSION Y(200),T(100)
        S(KL)=J
        IF (YJ.EQ.0) GO TO 210
        Y(J)=1
        W(J)=KL
        Z2FIX1=Z2FIX1+C2(J)
        GO TO 220
210    Y(J)=0
        W(J)=-KL

```

```

220 FEAS=0
    KSTART=COL(J)
    KSTOP=COL(J+1)-1
    DO 270 K=KSTART,KSTOP
        I=ROWNOS(K)
        IF (YJ.EQ.1) RHS(I)=RHS(I)-A(K)
        IF (A(K).LT.0) T(I)=T(I)-A(K)
        IF (T(I).GT.RHS(I)) FEAS=I
270 CONTINUE
280 RETURN
    END

```

```

C*****
C BEGIN SUBROUTINE RESET*
C*****

```

```

    SUBROUTINE RESET(KL,Y,W,V,S,Z2FIX1,T,RHS,COL,
1      ROWNOS,A,C2,NUMXY)
C    THIS SUBROUTINE FREES VARIABLES WHICH HAVE BEEN
C    SET AT LEVELS HIGHER THAN KL
C
    IMPLICIT INTEGER (A-Z)
    DIMENSION COL(200),ROWNOS(800),A(800),C2(200)
    DIMENSION W(200),V(200),S(200),RHS(100)
    DIMENSION Y(200),T(100)
    DO 370 L=1,NUMXY
        JL=ABS(W(L))
        IF (JL.LT.KL) GO TO 370
        Y(L)=0
        V(JL)=0
        S(JL)=0
        IF (W(L).GT.0) Z2FIX1=Z2FIX1-C2(L)
        KSTART=COL(L)
        KSTOP=COL(L+1)-1
        DO 360 K=KSTART,KSTOP
            I=ROWNOS(K)
            IF (W(L).GT.0) RHS(I)=RHS(I)+A(K)
            IF (A(K).LT.0) T(I)=T(I)+A(K)
360    CONTINUE
        W(L)=0
370 CONTINUE
    RETURN
    END

```

```

C*****
C BEGIN SUBROUTINE FTEST*
C*****
      SUBROUTINE FTEST(COL,ROWNOS,A,RHS,T,W,V,KL,YJ,
1          FEAS,NUMXY,Z2FIX1,S,Y,C2,J)
C  THIS SUBROUTINE CHECKS FEASIBILITY CONDITIONS TO
C  SET VARIABLES
      IMPLICIT INTEGER (A-Z)
      DIMENSION COL(200),ROWNOS(800),A(800),C2(200)
      DIMENSION W(200),V(200),RHS(100),S(200)
      DIMENSION T(100),Y(200)
      DO 470 IJ=1,NUMXY
        IF (W(IJ).NE.0) GO TO 470
        KSTART=COL(IJ)
        KSTOP=COL(IJ+1)-1
        DO 420 K=KSTART,KSTOP
          I=ROWNOS(K)
          IF (A(K).EQ.0) GO TO 420
          IF (A(K).LT.0) GO TO 410
          IF ((RHS(I)-A(K)).GE.T(I)) GO TO 420
          KL=KL+1
          V(KL)=1
          J=IJ
          YJ=0
          GO TO 415
410        IF (RHS(I).GE.(T(I)-A(K))) GO TO 420
          KL=KL+1
          V(KL)=1
          J=IJ
          YJ=1
415        CALL SET(Z2FIX1,KL,W,S,Y,T,RHS,COL,ROWNOS,A,
1              C2,FEAS,YJ,J)
          IF (FEAS.NE.0) GO TO 480
          GO TO 470
420      CONTINUE
470 CONTINUE
480 RETURN
      END

```


A.4 XMP Variable Definitions

The algorithms which solve the BLPP and the mixed integer BLPP rely on the XMP library of linear programming subroutines (Marsten 1981). Because of this, many of the variables used in the two algorithms are variables required by XMP. In this section, the XMP variables used in the two algorithms are defined.

DICTIONARY OF XMP VARIABLES

B IS THE RIGHT-HAND SIDE ARRAY.
 BASCB IS AN ARRAY CONTAINING THE OBJECTIVE
 FUNCTION COEFFICIENTS OF THE BASIC
 VARIABLES.
 BASIS IS THE LIST OF BASIC VARIABLES.
 BASLB IS AN ARRAY CONTAINING THE LOWER BOUNDS
 ON THE BASIC VARIABLES.
 BASUB IS AN ARRAY CONTAINING THE UPPER BOUNDS
 ON THE BASIC VARIABLES.
 BLOW CONTAINS THE LOWER LIMIT FOR EACH
 TWO-SIDED CONSTRAINT. (UPPER LIMIT
 IS IN B).
 BNDTYP 1 MEANS LOWER BOUND=0, UPPER
 BOUND=+INFINITY FOR EVERY NON-FREE
 VARIABLE;
 2 MEANS LOWER BOUND=0, UPPER BOUND=BOUND
 FOR EVERY NON-FREE STRUCTURAL VARIABLE
 3 MEANS LOWER BOUND=0 FOR EVERY NON-FREE
 VARIABLE;
 4 MEANS BOTH BOUNDS ARE GENERAL.
 BOUND IS THE COMMON UPPER BOUND WHEN BNDTYP=2.
 CAND IS THE CANDIDATE LIST; IT CONTAINS THE
 NON-BASIC VARIABLES THAT ARE ELIGIBLE TO
 ENTER THE BASIS DURING A SERIES OF MINOR
 ITERATIONS.
 CANDA IS A TABLE CONTAINING THE NON-ZERO
 COEFFICIENTS OF THE COLUMNS THAT BELONG
 TO THE CANDIDATE LIST.
 CANDCJ IS A LIST CONTAINING THE OBJECTIVE
 FUNCTION COEFFICIENT FOR EACH COLUMN
 THAT BELONGS TO THE CANDIDATE LIST.
 CANDI IS A TABLE CONTAINING THE ROW NUMBERS
 CORRESPONDING TO THE NON-ZEROS OF THE
 COLUMNS THAT BELONG TO THE CANDIDATE
 LIST.
 CANDL IS A LIST CONTAINING THE NUMBER OF
 NON-ZEROS IN EACH COLUMN THAT BELONGS
 TO THE CANDIDATE LIST.
 CJ IS USED TO HOLD THE OBJECTIVE FUNCTION
 COEFFICIENT.
 COLA IS USED TO HOLD THE NON-ZERO COEFFICIENTS
 OF A MATRIX COLUMN.

C	COLI	IS USED TO HOLD THE ROW NUMBERS
C		CORRESPONDING TO THE NON-ZEROS OF A
C		MATRIX COLUMN.
C	COLLEN	IS USED TO HOLD THE ROW NUMBERS
C		CORRESPONDING TO THE NON-ZEROS OF A
C		MATRIX COLUMN.
C	COLMAX	IS THE MAXIMUM NUMBER OF NON-ZEROS IN ANY
C		MATRIX COLUMN.
C	FACTOR	IS THE REFACTORIZATION FREQUENCY.
C	IOERR	IS THE I/O UNIT WHERE ERROR MESSAGES
C		ARE TO BE WRITTEN.
C	IOLOG	IS THE I/O UNIT WHERE LOG INFORMATION
C		IS TO BE WRITTEN, IF REQUESTED.
C	ITER	IS THE NUMBER OF ITERATIONS PERFORMED.
C	ITER1	CONTAINS THE NUMBER OF PHASE 1 ITERATIONS
C		PERFORMED.
C	ITER2	CONTAINS THE NUMBER OF PHASE 2 ITERATIONS
C		PERFORMED.
C	LENI	LENGTH OF THE WORKING STORAGE ARRAYS FOR
C		INTEGER DATA.
C	LENR	LENGTH OF THE WORKING STORAGE ARRAYS FOR
C		REAL DATA.
C	LENMI	LENGTH OF THE MAP OF THE WORKING STORAGE
C		ARRAY FOR INTEGER DATA.
C	LENMR	LENGTH OF THE MAP OF THE WORKING STORAGE
C		ARRAY FOR REAL DATA.
C	LJ	IS USED TO HOLD THE LOWER BOUND ON
C		A SINGLE VARIABLE.
C	LOOK	IS THE NUMBER OF MATRIX COLUMNS TO BE
C		CONSIDERED DURING CONSTRUCTION OF THE
C		CANDIDATE LIST.
C	M	IS THE NUMBER OF CONSTRAINTS.
C	MAPI	IS A MAP OF ARRAY MEMI.
C	MAPR	IS A MAP OF ARRAY MEMR.
C	MAXA	IS THE MAXIMUM NUMBER OF NON-ZEROS THAT
C		WILL BE ENCOUNTERED IN THE CONSTRAINT
C		MATRIX DURING THE CURRENT RUN.
C	MAXM	IS THE MAXIMUM NUMBER OF CONSTRAINTS THAT
C		WILL BE ENCOUNTERED DURING THE CURRENT
C		RUN; IT IS USED TO SET ARRAY SIZES.
C	MAXN	IS THE MAXIMUM NUMBER OF VARIABLES THAT
C		WILL BE ENCOUNTERED DURING THE CURRENT
C		RUN; IT IS USED TO SET ARRAY SIZES.
C	MEMI	ARRAY CONTAINING HIDDEN INTEGER DATA.
C	MEMR	ARRAY CONTAINING HIDDEN REAL DATA.

C N IS THE NUMBER OF VARIABLES (TOTAL,
 C INCLUDING SLACKS AND ARTIFICIALS).
 C NTYPE2 IF BNDTYP=2, THEN EACH VARIABLE
 C 1,2,...,NTYPE2 MUST EITHER SHARE THE
 C COMMON UPPER BOUND OR ELSE BE A FREE
 C VARIABLE. EACH OF THE REMAINING
 C VARIABLES NTYPE2+1,...,N MUST EITHER
 C HAVE LOWER BOUND ZERO AND UPPER BOUND
 C +INFINITY OR ELSE BE A FREE VARIABLE.
 C PICK IS THE SIZE OF THE CANDIDATE LIST USED
 C FOR MULTIPLE PRICING.
 C PRINT SPECIFIES THE LEVEL OF PRINTING DESIRED:
 C 0 MEANS ERROR MESSAGES ONLY;
 C 1 MEANS TERMINATION CONDITION MESSAGES;
 C 2 MEANS PRINT OBJECTIVE FUNCTION VALUE
 C AFTER EACH BASIS RE-FACTORIZATION;
 C 3 MEANS LOG INFORMATION AT EVERY
 C ITERATION.
 C ROWTYP IS THE ARRAY OF ROW TYPES:
 C +2 MEANS A TWO-SIDED CONSTRAINT;
 C +1 MEANS LESS THAN OR EQUAL TO;
 C 0 MEANS EQUATION;
 C -1 MEANS GREATER THAN OR EQUAL TO;
 C -2 MEANS A FREE ROW (FUNCTIONAL).
 C SCODE IS A RETURN CODE FOR XSTART:
 C 1 MEANS EVERYTHING OK;
 C 2 MEANS THE GIVEN BASIS WAS SINGULAR.
 C STATUS AN INDICATOR FOR EACH VARIABLE:
 C 0 MEANS THE VARIABLE IS OUT AT ITS
 C LOWER BOUND;
 C K MEANS THAT THIS IS THE K-TH BASIC
 C VARIABLE;
 C -1 MEANS THAT THIS VARIABLE IS OUT AT ITS
 C UPPER BOUND;
 C -2 MEANS THAT THIS IS A FREE VARIABLE...
 C ONCE IN THE BASIS IT NEVER LEAVES;
 C -3 MEANS THAT THIS IS AN ARTIFICIAL
 C VARIABLE...ONCE IT LEAVES THE BASIS IT
 C NEVER RE-ENTERS.
 C -4 MEANS THE VARIABLE IS LOCKED OUT OF
 C THE BASIS AT ITS LOWER BOUND;
 C -5 MEANS THE VARIABLE IS LOCKED OUT OF
 C THE BASIS AT ITS UPPER BOUND.
 C NOTE: FREE AND ARTIFICIAL VARIABLES
 C ALWAYS HAVE STATUS -2 OR -3 RESPECTIVELY,

C THEY DO NOT GET A POSITIVE STATUS WHEN
C THEY ARE IN THE BASIS.
C NOTE: SUPER-BASIC VARIABLES HAVE POSITIVE
C STATUS GREATER THAN M.
C TERMIN IS THE TERMINATION CODE FOR THE PRIMAL
C SIMPLEX METHOD:
C 1 MEANS OPTIMAL SOLUTION FOUND;
C 2 MEANS PROBLEM IS UNBOUNDED;
C 3 MEANS THE PROBLEM IS INFEASIBLE;
C 4 MEANS THE PRESUMED OPTIMAL SOLUTION
C DOES NOT SATISFY THE ACCURACY CHECK;
C 5 MEANS THE PROBLEM HAS BEEN ABANDONED.
C UJ IS USED TO HOLD THE UPPER BOUND ON A
C SINGLE VARIABLE.
C UNBDDQ IF THE PROBLEM IS UNBOUNDED, THEN UNBDDQ
C IS INDEX OF THE VARIABLE THAT WAS ABOUT
C TO ENTER THE BASIS WHEN THE UNBOUNDED
C CONDITION WAS DETECTED.
C UZERO THE ARRAY CONTAINING THE VALUES OF THE
C DUAL VARIABLES.
C XBZERO THE ARRAY CONTAINING THE VALUES OF THE
C BASIC VARIABLES AND THE SUPER-BASIC
C VARIABLES.
C YQ IS USED TO HOLD THE UNPACKED COLUMN THAT
C IS ABOUT TO ENTER THE BASIS.
C Z IS THE VALUE OF THE OBJECTIVE FUNCTION.

A.5 Programs for the BLPP

Three programs were created to support the algorithm for the BLPP. As with the 0-1 BLPP algorithm, the three programs create randomly generated BLPPs, allow the input of a BLPP, and solve the BLPP.

A.5.1 Program for Creating Randomly Generated BLPPs

The following program creates randomly generated BLPPs. The program calls the function URAND which returns randomly generated numbers. URAND is presented in section A.3. The randomly generated problem is output in its equivalent Kuhn-Tucker formulation. The program requires seven inputs. They are

- the number of variables controlled by the leader;
- the number of variables controlled by the follower;
- the number of constraints with no variables controlled by the follower;
- the maximum number of vertices allowed in the branch and bound tree;
- the maximum number of nonzero elements in a column of the A or B matrix;
- the random number seed.

The program contains variable definitions and documentation.

PROGRAM MATGEN

C
 C *****PURPOSE
 C PROGRAM GENERATES RANDOM BILEVEL LINEAR
 C PROGRAMMING PROBLEMS (BLPP). THE FUNCTION URAND
 C PROVIDES RANDOM NUMBERS. REFER TO THE SECTION ON
 C THE FUNCTION URAND FOR MORE INFORMATION AND A
 C LISTING OF THE FUNCTION.
 C
 C ***VARIABLE DEFINITIONS***
 C
 C ACC ARRAY WHICH CONTAINS THE COEFFICIENTS OF
 C EACH VARIABLE IN THE A- AND B-MATRIX OF
 C OF THE BLPP.
 C B THE RIGHT HAND SIDE ARRAY.
 C BNDTYP 1 MEANS LOWER BOUND=0, UPPER
 C BOUND=+INFINITY
 C FOR EVERY NON-FREE VARIABLE;
 C 2 MEANS LOWER BOUND=0, UPPER BOUND=BOUND
 C FOR EVERY NON-FREE STRUCTURAL VARIABLE
 C 3 MEANS LOWER BOUND=0 FOR EVERY NON-FREE
 C VARIABLE;
 C 4 MEANS BOTH BOUNDS ARE GENERAL.
 C CJ1 ARRAY CONTAINING THE LEADER'S OBJECTIVE
 C FUNCTION COEFFICIENTS.
 C CJ2 ARRAY CONTAINING THE FOLLOWER'S OBJECTIVE
 C FUNCTION COEFFICIENTS.
 C COLA IS USED TO HOLD THE NON-ZERO COEFFICIENTS
 C OF A MATRIX COLUMN.
 C COLI IS USED TO HOLD THE ROW NUMBERS
 C CORRESPONDING TO THE NON-ZEROS OF A MATRIX
 C COLUMN.
 C COLLEN THE NUMBER OF NONZEROS IN EACH COLUMN OF
 C THE A- AND B-MATRIX OF THE BLPP.
 C CONX NUMBER OF CONSTRAINTS CONTAINING NO
 C VARIABLES CONTROLLED BY THE FOLLOWER.
 C CONY NUMBER OF CONSTRAINTS CONTAINING VARIABLES
 C CONTROLLED BY THE FOLLOWER.
 C CRITV NUMBER OF COMPLEMENTARY SLACKNESS
 C CONDITIONS WHICH MUST BE SATISFIED IN
 C SOLVING THE BLPP IN ITS EQUIVALENT KUHN-
 C TUCKER REPRESENTATION.
 C IY RANDOM NUMBER SEED OF IY.

```

C      LJ      IS USED TO HOLD THE LOWER BOUND ON A
C              SINGLE VARIABLE.
C      M        NUMBER OF CONSTRAINTS IN THE EQUIVALENT
C              KUHN-TUCKER REPRESENTATION OF THE BLPP.
C      MAXCOL   MAXIMUM NUMBER OF NONZEROS IN ANY ONE
C              COLUMN OF THE BLPP.
C      MINMAX   FLAG INDICATING NATURE OF THE PROBLEM.
C              1 MEANS MAXIMIZATION; -1 MEANS
C              MINIMIZATION. FIXED AT 1 IN THIS PROGRAM.
C      NSTRUC   NUMBER OF VARIABLES IN THE KUHN-TUCKER
C              REPRESENTATION OF THE BLPP.
C      NUMCON   THE NUMBER OF CONSTRAINTS IN THE BLPP.
C      NUMITR   THE MAXIMUM NUMBER OF VERTICES IN THE
C              BRANCH AND BOUND TREE CREATED WHEN SOLVING
C              THE BLPP.
C      NUMX     THE NUMBER OF VARIABLES CONTROLLED BY THE
C              LEADER.
C      NUMXY    THE NUMBER OF VARIABLES IN THE BLPP.
C      NUMY     THE NUMBER OF VARIABLES CONTROLLED BY THE
C              FOLLOWER.
C      ROWTYP   ARRAY WHICH INDICATES THE TYPE OF
C              CONSTRAINT:
C              +1 MEANS LESS THAN OR EQUAL TO;
C              0 MEANS EQUATION;
C              -1 MEANS GREATER THAN OR EQUAL TO.
C      UJ       IS USED TO HOLD THE UPPER BOUND ON A
C              SINGLE VARIABLE.
C      UROW     ARRAY CONTAINS VALUES USED TO CREATE THE
C              KUHN-TUCKER STATIONARITY CONDITIONS OF THE
C              FOLLOWER'S PROBLEM.
C      ***END VARIABLE DEFINITIONS***
C
C      ***NOTE***
C      X REFERS TO VARIABLES CONTROLLED BY THE LEADER;
C      Y REFERS TO VARIABLES CONTROLLED BY THE FOLLOWER;
C      U REFERS TO KUHN-TUCKER MULTIPLIERS ASSOCIATED
C      WITH CONSTRAINTS CONTAINING VARIABLES
C      CONTROLLED BY THE FOLLOWER AND WITH THE
C      NONNEGATIVE RESTRICTIONS ON THE VARIABLES
C      CONTROLLED BY THE FOLLOWER;
C      G REFERS TO THE SLACK OR SURPLUS VARIABLE
C      ASSOCIATED WITH EACH CONSTRAINT CONTAINING
C      VARIABLES CONTROLLED BY THE FOLLOWER.
C      ***END OF NOTE***
C

```



```

REAL UROW(200,200),COLA(200),B(200),ACC(200,200),
1    CJ1(200),CJ2(200),UJ,LJ
INTEGER CNT,COLI(200),COLLEN(200),CONY,CONX,CONXY,
1    CRITV,PRINT,ECHO,BNDTYP,ROWTYP(200)
CHARACTER*4 DUMY
AONE=1.0
IONE=1
ZERO=0.0
IZERO=0
DUMY='XSUB
READ (5,*) NUMX,NUMY,CONX,CONY,NUMITR,MAXCOL,IY
MINMAX=1
ECHO=0
PRINT=0
BNDTYP=4
UJ=200000.0
LJ=0.0
NUMXY=NUMX+NUMY
CRITV=NUMY+CONY
M=NUMY+CONY+CONX
NSTRUC=NUMY+NUMY+CONY+CONY+NUMX
CONX1=CONX+1
NUMX1=NUMX+1
CONXY=CONX+CONY
C    ***COMPUTE RHS OF CONSTRAINTS WITH NO VARIABLES
C    CONTROLLED BY THE FOLLOWER AND CONSTRAINTS
C    WITH VARIABLES CONTROLLED BY THE FOLLOWER***
DO 100 I=1,CONXY
    ROWTYP(I)=1
    DUMMY=URAND(IY)
    IF (DUMMY.LT.0.3) ROWTYP(I)=-1
    B(I)=DUMMY*50.0
100 CONTINUE
C    ***COMPUTE COEFFICIENTS OF LEADER AND FOLLOWER
C    OBJECTIVE FUNCTIONS***
DO 120 I=1,NUMXY
    CJ1(I)=URAND(IY)*40.0-20.0
    CJ2(I)=URAND(IY)*40.0-20.0
120 CONTINUE
MAXC2=MAXCOL+MAXCOL
C    ***GENERATE THE A- AND B-MATRICES OF BLPP***
DO 130 I=1,NUMXY
    IROW=0
    CNT=0
    IF (I.GE.NUMX1) IROW=CONX

```

```

DO 140 J=1,MAXC2,2
  YA=URAND(IY)*60.0-15.0
  ROWNUM=URAND(IY)*3.7+1.0
  IROW=ROWNUM+IROW
  IF(IROW.GT.CONXY) GO TO 145
  ACC(I,J)=IROW
  ACC(I,J+1)=YA
  UROW(I,IROW)=YA
  CNT=CNT+1
140  CONTINUE
145  COLLEN(I)=CNT
130  CONTINUE
C    ***WRITE PROBLEM INFORMATION CONSTANTS***
    WRITE (6,905) CRITV,NUMX,NUMY,CONX,CONY,NUMITR
905  FORMAT (6I5)
    WRITE (6,908) M,NSTRUC,MINMAX,ECHO,PRINT,BNDTYP
908  FORMAT(6I5)
C    ***WRITED ROWTYP AND RHS OF CONSTRAINTS***
    DO 150 I=1,CONX
      WRITE (6,910) I,ROWTYP(I),B(I)
150  CONTINUE
    DO 151 I=CONX1,CONXY
      WRITE (6,910) I,IZERO,B(I)
151  CONTINUE
910  FORMAT(2I5,F13.3)
C    ***WRITE ROWTYP AND RHS OF U-ROWS***
    DO 160 I=1,NUMY
      K=I+CONXY
      J=I+NUMX
      WRITE (6,910) K,IZERO,-CJ2(J)
160  CONTINUE
C    ***WRITE X VARIABLE INFO AND COLUMNS***
    DO 230 I=1,NUMX
      ICOLL=COLLEN(I)
      WRITE (6,915) DUMY,I,ICOLL,CJ1(I),UJ,LJ,IZERO
      ICOLL2=2*ICOLL
      DO 235 J=1,ICOLL2,2
        IB=ACC(I,J)
234      WRITE (6,920) IB,ACC(I,J+1)
235      CONTINUE
230  CONTINUE
915  FORMAT(A4,2I5,F15.5,2F10.0,I2)
920  FORMAT(I5,F15.5)
C    ***WRITE Y VARIABLE INFO AND COLUMNS***
    DO 170 I=NUMX1,NUMXY

```

```

        ICOLL=COLLEN(I)
        ICOLL2=2*ICOLL
        WRITE (6,915) DUMY,I,ICOLL,CJ1(I),UJ,LJ,IZERO
        DO 180 J=1,ICOLL2,2
            IB=ACC(I,J)
            WRITE (6,920) IB,ACC(I,J+1)
180     CONTINUE
170 CONTINUE
C     ***WRITE G'S OF Y CONSTRAINTS***
        DO 190 I=1,CONY
            ISEQ=NUMXY+I
            WRITE (6,915) DUMY,ISEQ,IONE,ZERO,UJ,LJ,IZERO
            IDUM=I+CONX
            IF (ROWTYP(IDUM).EQ.-1) AONE=-1.0
            WRITE (6,920) IDUM,AONE
            AONE=1.0
190 CONTINUE
C     ***WRITE THE U'S OF THE Y .GE. 0 CONSTRAINTS***
        DO 200 I=1,NUMY
            ISEQ=NUMXY+CONY+I
            IROW=I+CONXY
            WRITE (6,915) DUMY,ISEQ,IONE,ZERO,UJ,LJ,IZERO
            WRITE (6,920) IROW,AONE
200 CONTINUE
C     ***WRITE THE U'S WHICH ARE KUHN-TUCKER***
C     MULTIPLIERS OF Y CONSTRAINTS DERIVED
C     FROM EQUATION
C     GRAD(F(Y))-U*GRAD(G(Y))=0***
        DO 210 I=CONX1,CONXY
            CNT=0
            DO 215 J=NUMX1,NUMXY
                IF(UROW(J,I).EQ.0.0) GO TO 215
                CNT=CNT+1
                COLA(CNT)=-UROW(J,I)
                IF (ROWTYP(I).EQ.-1) COLA(CNT)=UROW(J,I)
                COLI(CNT)=J+CONXY-NUMX
215     CONTINUE
                ISEQ=I+CONY+NUMY+NUMXY-CONX
                WRITE (6,915) DUMY,ISEQ,CNT,ZERO,UJ,LJ,IZERO
                DO 220 K=1,CNT
                    WRITE (6,920) COLI(K),COLA(K)
220     CONTINUE
210 CONTINUE
        STOP
        END

```

A.5.2 Program which Accepts Inputs of a BLPP and
Creates its Equivalent Kuhn-Tucker
Representation

The following program has two functions. It can be used to input the data of a BLPP or a mixed integer BLPP. It then creates a formulation of the data which can be used as input to the program which solves the BLPP or mixed integer BLPP. The instructions for inputting the data are contained in the program.

PROGRAM BLPPRD

C
 C *****PURPOSE
 C THIS PROGRAM READS IN A BILEVEL PROGRAMMING
 C PROBLEM (BLPP) OR A MIXED INTEGER BLPP AND CREATES
 C THE MATHEMATICAL PROGRAMMING PROBLEM WHICH CAN BE
 C SOLVED USING THE ALGORITHMS PRESENTED IN CHAPTER 5
 C (BLPP) OR CHAPTER 6 (MIXED INTEGER BLPP) TO SOLVE
 C THE PROBLEM.
 C
 C *****DIRECTIONS*****
 C TO CREATE A REPRESENTATION OF THE BLPP OR MIXED
 C INTEGER BLPP WHICH CAN BE SOLVED BY THE
 C APPROPRIATE ALGORITHM, ENTER THE DATA OF THE BLPP
 C OR MIXED INTEGER BLPP AS OUTLINED BELOW.
 C
 C THE CONSTRAINTS MUST BE IN A CERTAIN ORDER.
 C EXAMINE THE CONSTRAINTS AND PUT THOSE WHICH HAVE
 C NO VARIABLES CONTROLLED BY THE FOLLOWER BEFORE
 C THOSE WHICH DO.
 C
 C THE VARIABLES MUST ALSO BE PLACED IN A SPECIFIC
 C ORDER. THE VARIABLES CONTROLLED BY THE LEADER
 C MUST PRECEDE THOSE CONTROLLED BY THE FOLLOWER.
 C
 C ALL DATA IS INPUT USING FREE FORMAT. THIS MEANS
 C DATA ON A CARD ARE SEPARATED BY COMMAS OR SPACES.
 C EACH "CARD" OR LINE OF DATA REPRESENTS AN INPUT
 C FIELD.
 C
 C THE FIRST LINE OF DATA HAS FOUR ENTRIES
 C THE FOUR INTEGER ELEMENTS ENTERED ARE:
 C ENTRY 1: PROBLEM TYPE: +1 MEANS MAXIMIZE;-1
 C MEANS MINIMIZE;
 C ENTRY 2: INPUT DATA ECHO: 1 MEANS ECHO DATA;
 C 0 MEANS NO ECHO;
 C ENTRY 3: XMP PRINT SPECIFICATION. 0 IS BEST.
 C SEE THE XMP DICTIONARY FOR A DEFINITION
 C OF THE VARIOUS PRINT OPTIONS;
 C ENTRY 4: 1--ALL VARIABLES REAL VALUED;
 C 2--SOME VARIABLES ARE INTEGER.
 C
 C THE NEXT LINE OF DATA GIVES THE BASICS OF THE
 C PROBLEM. ALL ENTRIES ARE INTEGER.
 C ENTRY 1: NUMBER OF CONSTRAINTS WITHOUT VARIABLES

C CONTROLLED BY THE FOLLOWER;
 C ENTRY 2: NUMBER OF CONSTRAINTS WITH VARIABLES
 C CONTROLLED BY THE FOLLOWER.
 C ENTRY 3: NUMBER OF VARIABLES CONTROLLED BY
 C THE LEADER;
 C ENTRY 4: NUMBER OF VARIABLES CONTROLLED BY
 C THE FOLLOWER;
 C ENTRY 5: MAXIMUM NUMBER OF VERTICES ALLOWED
 C IN THE BRANCH AND BOUND TREE.
 C
 C ENTER THE COEFFICIENT OF EACH VARIABLE IN THE
 C LEADER'S OBJECTIVE FUNCTION. EACH COEFFICIENT
 C GOES ON ONE LINE OF DATA, I.E. ONE ENTRY PER LINE.
 C IF A VARIABLE'S COEFFICIENT IS ZERO, ENTER 0.0.
 C ENTRY 1: THE COEFFICIENT OF EACH VARIABLE IN THE
 C LEADER'S OBJECTIVE FUNCTION.
 C
 C ENTER THE COEFFICIENTS OF THE LINEAR TERMS IN THE
 C FOLLOWER'S OBJECTIVE FUNCTION. ONLY ENTER THE
 C COEFFICIENT OF THOSE VARIABLES CONTROLLED BY THE
 C FOLLOWER. AGAIN, THERE IS ONLY ONE ENTRY PER
 C LINE IF A VARIABLE CONTROLLED BY THE FOLLOWER HAS
 C A ZERO COEFFICIENT IN THE FOLLOWER'S OBJECTIVE
 C FUNCTION ENTER 0.0 FOR THAT VARIABLE.
 C ENTRY 1: THE COEFFICIENT OF EACH VARIABLE
 C CONTROLLED BY THE FOLLOWER IN THE
 C FOLLOWER'S OBJECTIVE FUNCTION.
 C
 C ENTER INFORMATION ON EACH CONSTRAINT. THE
 C INFORMATION IS CONSTRAINT NUMBER, CONSTRAINT TYPE,
 C AND THE RIGHT-HAND-SIDE. THE CONSTRAINT TYPES ARE
 C AS FOLLOWS:
 C
 C +1--LESS THAN OR EQUAL CONSTRAINT;
 C 0--EQUALITY CONSTRAINT;
 C -1--GREATER THAN OR EQUAL CONSTRAINT.
 C
 C THE RIGHT-HAND SIDE ENTRY MUST BE POSITIVE.
 C THE FIRST TWO ENTRIES ARE INTEGER AND THE THIRD
 C IS REAL. THERE IS ONE LINE OF DATA PER
 C CONSTRAINT.
 C
 C ENTRY 1: ENTER THE CONSTRAINT NUMBER;
 C ENTRY 2: ENTER THE CONSTRAINT TYPE
 C (EITHER -1,0, OR +1);

C ENTRY 3: ENTER THE RIGHT-HAND-SIDE OF THE
 C CONSTRAINT(>0).
 C
 C ENTER THE INFORMATION ON EACH VARIABLE.
 C INFORMATION ABOUT THE VARIABLES CONTROLLED BY THE
 C LEADER IS ENTERED FIRST. INFORMATION ABOUT
 C VARIABLES CONTROLLED BY THE FOLLOWER IS ENTERED
 C AFTER THE INFORMATION ON THE VARIABLES CONTROLLED
 C BY THE LEADER. ENTER ALL DATA ON A VARIABLE
 C BEFORE PROCEEDING TO THE NEXT VARIABLE. A LINE
 C OF DATA ON EACH VARIABLE CONTAINS GENERAL
 C INFORMATION ABOUT THAT VARIABLE. THE THREE
 C ENTRIES ARE INTEGER WHERE:
 C
 C ENTRY 1: THE VARIABLE NUMBER;
 C ENTRY 2: NUMBER OF NON-ZERO COEFFICIENTS IN THE A
 C OR B MATRIX;
 C ENTRY 3: VARIABLE TYPE WHERE THE TYPES ARE AS
 C FOLLOWS:
 C 0--A REAL VALUED, UNBOUNDED VARIABLE;
 C 1--AN INTEGER, UNBOUNDED VARIABLE;
 C 2--AN UPPER BOUNDED, INTEGER VARIABLE;
 C 3--AN UPPER BOUNDED, REAL VALUED
 C VARIABLE.
 C
 C IF A VARIABLE HAS AN UPPER BOUND, THE NEXT LINE OF
 C DATA IS THE UPPER BOUND. IF THE VARIABLE IS NOT
 C UPPER BOUNDED, THE UPPER BOUND IS SET TO 200,000.
 C THE LOWER BOUND ON ALL VARIABLES IS SET TO 0.0.
 C
 C ENTER THE VARIABLE'S NON-ZERO COEFFICIENTS IN THE
 C A OR B MATRIX. TO ENTER THIS INFORMATION, TWO
 C DATA ENTRIES PER LINE ARE MADE. THE FIRST ENTRY
 C IS INTEGER AND THE SECOND IS REAL.
 C
 C ENTRY 1: THE ROW NUMBER (CONSTRAINT) IN WHICH THE
 C COEFFICIENT APPEARS;
 C ENTRY 2: THE COEFFICIENT.
 C
 C AFTER ALL NON-ZERO COEFFICIENTS OF A VARIABLE HAVE
 C BEEN ENTERED, GO TO THE NEXT VARIABLE AND ENTER
 C INFORMATION FOR IT.
 C
 C IF THE PROBLEM HAS NONLINEAR TERMS IN THE
 C FOLLOWER'S OBJECTIVE FUNCTION, THIS INFORMATION

C IS NOW ENTERED.

C
C THE LAST INFORMATION ENTERED IS THE COEFFICIENTS
C OF THE NONLINEAR TERMS, IF ANY, IN THE FOLLOWER'S
C OBJECTIVE FUNCTION. A WAY TO REPRESENT THIS DATA
C IS THE FOLLOWING FORM:

C T
C (X,Y) Q (X,Y) WHERE Q IS A LOWER TRIANGULAR
C MATRIX AND THE T MEANS THE TRANSPOSE OF THE ROW
C VECTOR (X,Y) WHERE X REPRESENTS THE VARIABLES
C CONTROLLED BY THE LEADER AND Y REPRESENTS THE
C VARIABLES CONTROLLED BY THE FOLLOWER. THIS ALLOWS
C THE REPRESENTATION OF EVERY POSSIBLE CROSS PRODUCT
C OF THE PROBLEM VARIABLES. THE ELEMENTS OF Q
C CORRESPOND TO THE COEFFICIENTS OF THESE TERMS IN
C THE FOLLOWER'S OBJECTIVE FUNCTION. FOR INSTANCE,
C THE ROW 1, COLUMN 1 ELEMENT OF Q IS THE
C COEFFICIENT OF THE PRODUCT OF VARIABLE ONE WITH
C VARIABLE ONE AND THE ROW 6, COLUMN 3 ELEMENT OF Q
C IS THE COEFFICIENT OF THE PRODUCT OF VARIABLE 3
C AND VARIABLE 6. SINCE Q IS LOWER TRIANGULAR,
C THERE IS NO DUPLICATION OF TERMS. IN THE INPUT
C DATA SET, ENTER A 1, 2, OR 3 WHERE THE NUMBERS
C HAVE THE FOLLOWING MEANING:

- C 1--NO NONLINEAR TERMS IN THE FOLLOWER'S
C OBJECTIVE FUNCTION;
C 2--THE Q MATRIX WILL BE ENTERED BY ROW,
C COLUMN, VALUE;
C 3--THE Q MATRIX WILL BE ENTERED A ROW AT
C A TIME.

C FURTHER EXPLANATION OF CHOICE 2--THREE ENTRIES ARE
C MADE. THE FIRST TWO ENTRIES ARE INTEGER AND THE
C THIRD IS REAL.

C ENTRY 1: THE ROW NUMBER;
C ENTRY 2: THE COLUMN NUMBER;
C ENTRY 3: THE COEFFICIENT OF THE NONLINEAR TERM.

C WHEN DATA ENTRY IS COMPLETE, ENTER 999,0,0 ON THE
C LAST LINE OF THE INPUT FILE.

C FURTHER EXPLANATION OF CHOICE 3--ENTER ONE REAL
C NUMBER PER LINE. ALL ELEMENTS OF Q ON OR BELOW
C THE DIAGONAL MUST BE ENTERED WITH ONE ELEMENT PER
C LINE. ENTER DATA BY ROWS AND INCLUDE ZEROS.


```

C *****END OF DIRECTIONS*****
C
C
C *****VARIABLE DEFINITIONS*****
C
C ACC      ARRAY WHICH CONTAINS THE COEFFICIENTS OF
C           EACH VARIABLE IN THE A- OR B-MATRIX OF THE
C           BLPP OR MIXED INTEGER BLPP.
C B        IS THE RIGHT HAND SIDE ARRAY.
C BNDTYP   1 MEANS LOWER BOUND=0, UPPER
C           BOUND=+INFINITY FOR EVERY NON-FREE
C           VARIABLE;
C           2 MEANS LOWER BOUND=0, UPPER BOUND=BOUND
C           FOR EVERY NON-FREE STRUCTURAL VARIABLE
C           3 MEANS LOWER BOUND=0 FOR EVERY NON-FREE
C           VARIABLE;
C           4 MEANS BOTH BOUNDS ARE GENERAL.
C           COEFFICIENT.
C BNONL    ARRAY CONTAINING THE COEFFICIENTS OF THE
C           NONLINEAR TERMS IN THE FOLLOWER'S
C           OBJECTIVE FUNCTION.
C CJ1      ARRAY CONTAINING THE LEADER'S OBJECTIVE
C           FUNCTION COEFFICIENTS.
C CJ2      ARRAY CONTAINING THE FOLLOWER'S OBJECTIVE
C           FUNCTION COEFFICIENTS.
C COLA     IS USED TO HOLD THE NON-ZERO COEFFICIENTS
C           OF A MATRIX COLUMN OF THE BLPP OR MIXED
C           INTEGER BLPP.
C COLI     IS USED TO HOLD THE ROW NUMBERS
C           CORRESPONDING TO THE NON-ZEROS OF A
C           MATRIX COLUMN OF THE BLPP OR MIXED
C           INTEGER BLPP.
C COLLEN   IS AN ARRAY USED TO HOLD THE NUMBER OF
C           NON-ZEROS IN A MATRIX COLUMN OF THE BLPP
C           OR MIXED INTEGER BLPP.
C CONX     NUMBER OF CONSTRAINTS CONTAINING NO
C           VARIABLES CONTROLLED BY THE FOLLOWER.
C CONY     NUMBER OF CONSTRAINTS CONTAINING VARIABLES
C           CONTROLLED BY THE FOLLOWER.
C CRITV    NUMBER OF COMPLEMENTARY SLACKNESS
C           CONDITIONS WHICH MUST BE SATISFIED IN
C           SOLVING THE BLPP IN ITS EQUIVALENT KUHN-
C           TUCKER REPRESENTATION. IN THE MIXED
C           INTEGER BLPP, CRITV IS USED IN SOLVING
C           THE RELAXED BLPP.

```

C ECHO FLAG USED BY SOLUTION ALGORITHMS.
 C 0 MEANS DO NOT ECHO INPUT DATA;
 C 1 MEANS ECHO INPUT DATA.
 C LJ IS USED TO HOLD THE LOWER BOUND ON
 C A SINGLE VARIABLE.
 C M IS THE NUMBER OF CONSTRAINTS.
 C MINMAX FLAG INDICATING NATURE OF THE PROBLEM.
 C 1 MEANS MAXIMIZATION AND -1 MEANS
 C MINIMIZATION.
 C MAXNDS THE MAXIMUM NUMBER OF VERTICES IN THE
 C BRANCH AND BOUND TREE CREATED WHEN SOLVING
 C THE BLPP OR MIXED INTEGER BLPP.
 C NPTYPE INDICATES THE TYPE OF PROBLEM. 1 MEANS
 C THE PROBLEM IS A BLPP. 2 MEANS THE
 C PROBLEM IS A MIXED INTEGER BLPP.
 C NSTRUC THE TOTAL NUMBER OF VARIABLES IN THE
 C KUHN-TUCKER REPRESENTATION OF THE BLPP OR
 C MIXED INTEGER BLPP.
 C NTERM INDICATES IF THE PROBLEM HAS NONLINEAR
 C TERMS IN THE FOLLOWER'S OBJECTIVE
 C FUNCTION.
 C 1 MEANS NO NONLINEAR TERMS IN THE
 C FOLLOWER'S OBJECTIVE FUNCTION;
 C 2 MEANS THE Q MATRIX WILL BE ENTERED BY
 C ROW, COLUMN, VALUE;
 C 3 MEANS THE Q MATRIX WILL BE ENTERED ONE
 C ROW AT A TIME.
 C NUMX THE NUMBER OF VARIABLES CONTROLLED BY THE
 C LEADER.
 C NUMXY THE NUMBER OF VARIABLES IN THE BLPP OR
 C MIXED INTEGER BLPP.
 C NUMY THE NUMBER OF VARIABLES CONTROLLED BY THE
 C FOLLOWER.
 C PRINT SPECIFIES THE LEVEL OF PRINTING DESIRED:
 C 0 MEANS ERROR MESSAGES ONLY;
 C 1 MEANS TERMINATION CONDITION MESSAGES;
 C 2 MEANS PRINT OBJECTIVE FUNCTION VALUE
 C AFTER EACH BASIS RE-FACTORIZATION;
 C 3 MEANS LOG INFORMATION AT EVERY
 C ITERATION.
 C ROWTYP ARRAY WHICH INDICATES THE TYPE OF
 C CONSTRAINT:
 C +1 MEANS LESS THAN OR EQUAL TO;
 C 0 MEANS EQUATION;
 C -1 MEANS GREATER THAN OR EQUAL TO.

```

C      UCOLA   IS USED TO HOLD THE NON-ZERO COEFFICIENTS
C              OF A MATRIX COLUMN OF ASSOCIATED WITH THE
C              KUHN-TUCKER MULTIPLIERS OF THE
C              STATIONARITY CONDITIONS.
C      UCOLI   IS USED TO HOLD THE ROW NUMBERS
C              CORRESPONDING TO THE NON-ZEROS OF A
C              MATRIX COLUMN OF THE STATIONARITY
C              CONDITIONS.
C      UCOLEN  IS AN ARRAY USED TO HOLD THE NUMBER OF
C              NON-ZEROS IN A MATRIX COLUMN OF THE
C              STATIONARITY CONDITIONS.
C      UJ      IS USED TO HOLD THE GENERAL UPPER BOUND
C              OF THE PROGRAM; UJ=200,000.
C              VARIABLE.
C      UJO     AN ARRAY WHICH STORES THE UPPER BOUND ON
C              THE VARIABLES CONTROLLED BY THE LEADER OR
C              THE FOLLOWER.
C      UROW    ARRAY CONTAINS VALUES USED TO CREATE THE
C              KUHN-TUCKER STATIONARITY CONDITIONS OF THE
C              FOLLOWER'S PROBLEM.
C      VARTYP  AN ARRAY CONTAINING THE VARIABLE TYPE
C              WHERE THE TYPES ARE AS FOLLOWS:
C              0--A REAL VALUED, UNBOUNDED VARIABLE;
C              1--AN INTEGER, UNBOUNDED VARIABLE;
C              2--AN UPPER BOUNDED, INTEGER VARIABLE;
C              3--AN UPPER BOUNDED, REAL VALUED
C              VARIABLE.
C      XINT    NUMBER OF INTEGER VARIABLES CONTROLLED BY
C              THE LEADER.
C      YINT    NUMBER OF INTEGER VARIABLES CONTROLLED BY
C              THE FOLLOWER.
C      *****END VARIABLE DEFINITIONS
C
C      *****NOTE*****
C      X REFERS TO VARIABLES CONTROLLED BY THE LEADER;
C      Y REFERS TO VARIABLES CONTROLLED BY THE FOLLOWER;
C      U REFERS TO KUHN-TUCKER MULTIPLIERS ASSOCIATED
C      WITH CONSTRAINTS CONTAINING VARIABLES
C      CONTROLLED BY THE FOLLOWER AND WITH THE
C      NONNEGATIVE RESTRICTIONS ON THE VARIABLES
C      CONTROLLED BY THE FOLLOWER;
C      G REFERS TO THE SLACK OR SURPLUS VARIABLE
C      ASSOCIATED WITH EACH CONSTRAINT CONTAINING
C      VARIABLES CONTROLLED BY THE FOLLOWER.
C      *****END OF NOTE*****

```

```

C
  REAL UROW(200,200),COLA(200),B(200),ACC(200,200),
1     CJ1(200),CJ2(200),UJ,LJ,UJO(200),
2     BNONL(200,200),UCOLA(400)
  INTEGER CNT,COLI(200),COLLEN(200),CONY,CONX,CONXY,
1     CRITV,PRINT,ECHO,BNDTYP,ROWTYP(200),
2     VARTYP(200),UCOLI(400),UCOLEN(200),
3     XINT,YINT
  CHARACTER*4 DUMY
  AONE=1.0
  IONE=1
  ZERO=0.0
  IZERO=0
  DUMY='XSUB'
  NSTRUC=0
C   ***READ PROBLEM INFORMATION***
  READ (5,*) MINMAX,ECHO,PRINT,NPTYPE
C
  READ (5,*) CONX,CONY,NUMX,NUMY,MAXNDS
C
  NUMXY=NUMX+NUMY
C   ***READ IN THE COEFFICIENTS OF THE VARIABLES IN
C   THE LEADER'S OBJECTIVE FUNCTION***
  DO 100 I=1,NUMXY
    READ (5,*) CJ1(I)
100  CONTINUE
C   ***READ IN THE COEFFICIENTS OF THE VARIABLES IN
C   THE FOLLOWER'S OBJECTIVE FUNCTION***
  DO 110 I=1,NUMY
    READ (5,*) CJ2(I+NUMX)
110  CONTINUE
C   ***READ IN CONSTRAINT TYPE AND RIGHT-HAND-SIDE FOR
C   EACH CONSTRAINT***
  CONXY=CONX+CONY
  DO 120 I=1,CONXY
    READ (5,*) J,ROWTYP(I),B(I)
    IF (ROWTYP(I).EQ.0) NSTRUC=NSTRUC+1
120  CONTINUE
C   ***SET UPPER BOUND FOR BLPP VARIABLES
C   AT 200,000***
  DO 125 I=1,NUMXY
    UJO(I)=200000.0
125  CONTINUE
C   ***READ IN VARIABLE INFORMATION AND NONZERO
C   COEFFICIENTS OF A- OR B-MATRIX***

```

```

XINT=0
YINT=0
DO 130 I=1,NUMXY
  READ (5,*) J,COLLEN(I),VARTYP(I)
  IF (VARTYP(I).EQ.0 .OR. VARTYP(I).EQ.3)
1    GO TO 126
  IF (I.LE.NUMX) XINT=XINT+1
  IF (I.GT.NUMX) YINT=YINT+1
126  IF (VARTYP(I).GE.2) READ (5,*) UJO(I)
  IF (VARTYP(I).GE.1 .AND. I.GT.NUMX)
1    NSTRUC=NSTRUC+1
  DO 140 K=1,COLLEN(I)
    KBY2=2*K
    READ (5,*) NUMROW,COEFF
    ACC(I,KBY2-1)=NUMROW
    ACC(I,KBY2)=COEFF
    UROW(I,NUMROW)=COEFF
140  CONTINUE
130  CONTINUE
C    ***READ IN COEFFICIENTS OF NONLINEAR TERMS IN THE
C    FOLLOWER'S OBJECTIVE FUNCTION***
  READ (5,*) NTERM
  GO TO(310,320,330) NTERM
320  READ (5,*) NROW,NCOL,VALUE
  IF (NROW.EQ.999) GO TO 310
  BNONL(NROW,NCOL)=VALUE.
  GO TO 320
330  DO 50 I=1,NUMXY
    DO 60 J=1,I
      READ (5,*) BNONL(I,J)
    60  CONTINUE
    50  CONTINUE
310  BNDTYP=4
    UJ=200000.0
    LJ=0.0
    CRITV=NUMY+CONY
    M=NUMY+CONY+CONX
    NSTRUC=NUMY+NUMY+CONY+CONY+NUMX+NSTRUC
    CONX1=CONX+1
    NUMX1=NUMX+1
C    ***WRITE PROBLEM INFORMATION***
    IF (NPTYPE.EQ.1)
1      WRITE (6,905) CRITV,NUMX,NUMY,CONX,CONY,MAXNDS
905  FORMAT (6I5)

```

```

      IF (NPTYPE.EQ.2)
1      WRITE (6,906) CRITV,NUMX,NUMY,CONX,CONY,XINT,
2      YINT,MAXNDS
906 FORMAT(8I5)
      WRITE (6,908) M,NSTRUC,MINMAX,ECHO,PRINT,BNDTYP
908 FORMAT(6I5)
C      ***WRITE INFORMATION ON CONSTRAINTS WITH NO
C      VARIABLES CONTROLLED BY THE FOLLOWER***
      DO 150 I=1,CONX
          WRITE (6,910) I,ROWTYP(I),B(I)
150 CONTINUE
910 FORMAT(2I5,F13.3)
C      ***WRITE INFORMATION ON CONSTRAINTS WITH VARIABLES
C      CONTROLLED BY THE FOLLOWER***
      DO 155 I=CONX1,CONXY
          WRITE (6,910) I,IZERO,B(I)
155 CONTINUE
C      ***WRITE RHS OF U-ROWS TO DATA SET (ISEQ, ROWTYP,
C      GRADIENT WITH RESPECT TO Y--LINEAR TERMS)***
      DO 160 I=1,NUMY
          K=I+CONXY
          J=I+NUMX
          WRITE (6,910) K,IZERO,-CJ2(J)
160 CONTINUE
C      ***WRITE THE TERMS TO THE GRADIENT ROWS WHICH
C      ARE DERIVED FROM THE NONLINEAR TERMS IN THE
C      FOLLOWER'S OBJECTIVE FUNCTION***
      IF (NTERM.EQ.1) GO TO 225
      DO 350 I=NUMX1,NUMXY
          IROW=CONXY+I-NUMX
          DO 360 J=1,I
              IF (BNONL(I,J).EQ.0.0) GO TO 360
              COLLEN(J)=COLLEN(J)+1
              K=COLLEN(J)*2
              ACC(J,K-1)=IROW
              ACC(J,K)=BNONL(I,J)
              IF (J.EQ.I) ACC(J,K)=2.0*BNONL(I,J)
              IF (J.LE.NUMX .OR. J.EQ.I) GO TO 360
              IROW1=CONXY+J-NUMX
              COLLEN(I)=COLLEN(I)+1
              K=2*COLLEN(I)
              ACC(I,K-1)=IROW1
              ACC(I,K)=BNONL(I,J)
360 CONTINUE

```

```

350 CONTINUE
C   ***WRITE X VARIABLE INFO AND COLUMNS***
225 DO 230 I=1,NUMX
      ICOLL=COLLEN(I)
      WRITE (6,915) DUMY,I,ICOLL,CJ1(I),
1         UJO(I),LJ,VARTYP(I)
      ICOLL2=2*ICOLL
      DO 235 J=1,ICOLL2,2
        IB=ACC(I,J)
234      WRITE (6,920) IB,ACC(I,J+1)
235      CONTINUE
230 CONTINUE
C   ***WRITE Y VARIABLE INFO AND COLUMNS TO DATA
C   SET***
      DO 170 I=NUMX1,NUMXY
        ICOLL=COLLEN(I)
        ICOLL2=2*ICOLL
        ISEQ=I
        WRITE (6,915) DUMY,ISEQ,ICOLL,CJ1(I),
1         UJO(I),LJ,VARTYP(I)
        DO 180 J=1,ICOLL2,2
          IB=ACC(I,J)
          WRITE (6,920) IB,ACC(I,J+1)
180      CONTINUE
170 CONTINUE
C   ***WRITE G'S OF CONSTRAINTS WITH VARIABLES
C   CONTROLLED BY THE FOLLOWER***
      DO 190 I=1,CONY
        IDUM=I+CONX
        ISEQ=NUMXY+I
        IF (ROWTYP(IDUM).EQ.0) UJ=0.0
        WRITE (6,915) DUMY,ISEQ,IONE,ZERO,UJ,LJ,IZERO
        UJ=200000.0
        IF (ROWTYP(IDUM).EQ.-1) AONE=-1.0
        WRITE (6,920) IDUM,AONE
        AONE=1.0
190 CONTINUE
C   ***WRITE THE U'S OF THE Y .GE. 0 CONSTRAINTS***
      DO 200 I=1,NUMY
        ISEQ=NUMXY+CONY+I
        IROW=I+CONXY
        WRITE (6,915) DUMY,ISEQ,IONE,ZERO,UJ,LJ,IZERO
        WRITE (6,920) IROW,AONE
200 CONTINUE
C   ***WRITE THE U'S WHICH ARE THE KUHN-TUCKER

```

```

C      MULTIPLIERS OF CONSTRAINTS WITH VARIABLES
C      CONTROLLED BY THE FOLLOWER AND ARE DERIVED
C      FROM THE EQUATION:
C      GRAD(F(Y))-U*GRAD(G(Y))=0***
      LENCNT=0
      DO 210 I=CONX1,CONXY
        CNT=0
        DO 215 J=NUMX1,NUMXY
          IF(UROW(J,I).EQ.0.0) GO TO 215
          CNT=CNT+1
          COLA(CNT)=-UROW(J,I)
          IF (ROWTYP(I).EQ.-1) COLA(CNT)=UROW(J,I)
          COLI(CNT)=J+CONXY-NUMX
215    CONTINUE
      ISEQ=I+CONY+NUMY+NUMY+NUMX-CONX
      WRITE (6,915) DUMY,ISEQ,CNT,ZERO,UJ,LJ,IZERO
      IF (ROWTYP(I).NE.0) GO TO 650
      UCOLEN(I)=CNT
      LENCNT=LENCNT+CNT
      KSTART=LENCNT-CNT+1
      KPNT=1
      DO 610 K=KSTART,LENCNT
        UCOLI(K)=COLI(KPNT)
        UCOLA(K)=-COLA(KPNT)
        WRITE (6,920) COLI(KPNT),COLA(KPNT)
        KPNT=KPNT+1
610    CONTINUE
      GO TO 210
650    DO 220 K=1,CNT
        WRITE (6,920) COLI(K),COLA(K)
220    CONTINUE
210  CONTINUE
C      ***WRITE U'S WHICH ARE THE KUHN-TUCKER MULTIPLIERS
C      ASSOCIATED WITH UPPER BOUND CONSTRAINTS ON Y***
      ISEQ=CONY+CONY+NUMY+NUMY+NUMX
      CNT=0
      DO 400 I=NUMX1,NUMXY
        CNT=CNT+1
        IF (VARTYP(I).EQ.0) GO TO 400
        ISEQ=ISEQ+1
        WRITE (6,915) DUMY,ISEQ,IONE,ZERO,UJ,LJ,IZERO
        IROW=CONXY+CNT
        WRITE (6,920) IROW,-AONE
400  CONTINUE
C      ***WRITE U'S WHICH ARE THE KUHN-TUCKER MULTIPLIERS

```



```
C      FOR THE EQUALITY CONSTRAINTS WITH VARIABLES
C      CONTROLLED BY THE FOLLOWER***
      CNT=0
      DO 500 I=CONX1,CONXY
        IF (ROWTYP(I).NE.0) GO TO 500
        CNT=CNT+UCOLEN(I)
        ISEQ=ISEQ+1
        WRITE (6,915) DUMY,ISEQ,UCOLEN(I),
1          ZERO,UJ,LJ,IZERO
        KSTART=CNT-UCOLEN(I)+1
        DO 510 K=KSTART,CNT
          WRITE (6,920) UCOLI(K),UCOLA(K)
510      CONTINUE
500 CONTINUE
915  FORMAT(A4,2I5,F15.5,2F10.0,I2)
920  FORMAT(I5,F15.5)
      STOP
      END
```

A.5.3. The BLPP Algorithm

The algorithm for the BLPP is computerized by the following program. The program solves problems created using the random problem generator of section A.5.1 or problems which were input using the code of section A.5.2. For definitions of the XMP variables, refer to section A.4.

PROGRAM BLPP

PURPOSE

THIS PROGRAM SOLVES THE LINEAR BLPP WHERE ALL VARIABLES HAVE POSITIVE REAL VALUES.

THE PROGRAM READS IN THE LINEAR PROBLEM WHICH RESULTS WHEN THE KUHN-TUCKER APPROACH IS USED TO REWRITE THE BLPP AS A STANDARD LINEAR PROGRAMMING PROBLEM. A BRANCH AND BOUND SCHEME WHICH FORCES SATISFACTION OF THE COMPLIMENTARY SLACKNESS CONDITIONS THROUGH MANIPULATION OF THE KUHN-TUCKER MULTIPLIERS AND THEIR ASSOCIATED EQUATIONS IS USED TO FIND THE OPTIMAL SOLUTION OF THE BLPP. THE XMP LIBRARY OF LINEAR PROGRAMMING SUBROUTINES IS USED TO SOLVE THE LINEAR PROGRAMMING SUBPROBLEM AT EACH VERTEX OF THE BRANCH AND BOUND TREE.

THIS PROGRAM READS IN THE PROBLEM DATA AND PASSES IT TO XMP WHICH SOLVES THE LINEAR PROGRAM BY THE PRIMAL SIMPLEX METHOD. THE SOLUTION IS USED TO SELECT A VARIABLE ON WHICH TO BRANCH. WHEN ALL LIVE VERTICES IN THE BRANCH AND TREE HAVE BEEN FATHOMED, THE INCUMBENT SOLUTION IS DECLARED OPTIMAL.

THE PROGRAM DOES NOT TAKE ANY ADVANTAGE OF SPECIAL PROBLEM TYPE OR STRUCTURE. FOR MOST SERIOUS APPLICATIONS YOU WILL WANT TO WRITE A MATRIX GENERATOR THAT READS IN DATA TABLES AND GENERATES THE LP COEFFICIENT MATRIX OF THE EQUIVALENT KUHN-TUCKER FORMULATION OF THE BLPP PROBLEM. THE PROGRAM BLPPRD CAN BE USED TO DO THIS.

DICTIONARY OF VARIABLES

XMP VARIABLES--FOR DEFINITIONS SEE THE
XMP DICTIONARY

VARIABLES USED IN BLPP BRANCH AND BOUND

BBTBIG CONTAINS VALUE OF BIGGEST PRODUCT IN TEST
FOR SELECTION OF THE BRANCHING VARIABLE.

C	BBTEMP	CONTAINS VALUE OF PRODUCT IN TEST
C		FOR SELECTION OF THE BRANCHING VARIABLE.
C	BBVAR	AN ARRAY WHICH CONTAINS THE VALUES OF
C		THE VARIABLES AT THE CURRENT VERTEX OF
C		THE BRANCH AND BOUND TREE.
C	BESTAT	IS THE STATUS OF EACH VARIABLE IN THE
C		INCUMBENT SOLUTION. REFER TO THE XMP
C		DICTIONARY FOR A DEFINITION OF STATUS.
C	BESTB	IS THE LIST OF BASIC VARIABLES IN THE
C		INCUMBENT SOLUTION.
C	BESTLB	ARRAY WHICH CONTAINS THE LOWER BOUNDS ON
C		THE VARIABLES IN THE INCUMBENT SOLUTION.
C	BESTUB	ARRAY WHICH CONTAINS THE UPPER BOUNDS ON
C		THE VARIABLES IN THE INCUMBENT SOLUTION.
C	BLVAR	POINTS TO POSITION WHERE KUHN-TUCKER
C		MULTIPLIERS OF UPPER BOUND ON VARTYP = 3
C		VARIABLES IS FOUND IN BBVAR.
C	CJ1	ARRAY WHICH STORES THE COEFFICIENTS OF THE
C		LEADER'S OBJECTIVE FUNCTION.
C	CONX	NUMBER OF CONSTRAINTS WHICH DO NOT CONTAIN
C		ANY VARIABLES CONTROLLED BY THE FOLLOWER.
C	CONY	NUMBER OF CONSTRAINTS WHICH CONTAIN
C		VARIABLES CONTROLLED BY THE FOLLOWER.
C	CRITV	IS THE NUMBER OF COMPLEMENTARY SLACKNESS
C		CONDITIONS WHICH MUST BE SATISFIED BY
C		BY THE SOLUTION OF THE BLPP.
C	DUMCNT	USED TO KEEP TRACK OF WHICH UPPER BOUNDED
C		VARIABLE IS BRANCHED ON AT CURRENT VERTEX.
C	IVARU	USED TO KEEP TRACK OF WHICH UPPER BOUNDED
C		VARIABLE IS BRANCHED ON AT EACH VERTEX.
C	LEVEL	CONTAINS THE LEVEL OF THE CURRENT VERTEX
C		OF THE BLPP BRANCH AND BOUND TREE.
C	NODE	COUNTS NUMBER OF VERTICES IN THE BRANCH
C		AND BOUND TREE
C	NODBST	STORE VERTEX AT WHICH INCUMBENT SOLUTION
C		FOUND.
C	NSTOPV	COUNTER USED IN LOOP USED TO SELECT THE
C		VARIABLE ON WHICH TO BRANCH.
C	NSTRUC	NUMBER OF VARIABLES IN THE KUHN-TUCKER
C		REPRESENTATION OF THE BLPP.
C	NUMITR	MAXIMUM NUMBER OF S IN THE BRANCH AND
C		BOUND TREE.
C	NUMX	NUMBER OF VARIABLES CONTROLLED BY THE
C		LEADER.
C	NUMY	NUMBER OF VARIABLES CONTROLLED BY THE

```

C      FOLLOWER.
C      OKBAS  ARRAY WHICH STORES THE MOST CURRENT
C              FEASIBLE BASIS.
C      OKSTAT ARRAY WHICH STORES STATUS OF EACH VARIABLE
C              IN THE MOST CURRENT FEASIBLE BASIS.
C      UJO    ARRAY WHICH STORES THE UPPER BOUNDS OF
C              VARIABLES WHICH HAVE UPPER BOUNDS.
C      VARTYP INDICATES THE VARIABLE TYPE.
C              0 MEANS A VARIABLE IS REAL VALUED;
C              1 MEANS A VARIABLE IS INTEGER VALUED;
C              2 MEANS A VARIABLE HAS AN UPPER BOUND AND
C                IS INTEGER VALUED;
C              3 MEANS A VARIABLE HAS AN UPPER BOUND AND
C                AND IS REAL VALUED.
C      NOTE:  ONLY VARTYPS OF 0 OR 3 ARE ALLOWED
C              THIS ALGORITHM.
C      VCNT   A COUNT ON NUMBER OF UPPER BOUNDED
C              VARIABLES (VARTYP = 3).
C      W      IS AN ARRAY WHICH INDICATES WHICH
C              COMPLEMENTARY SLACKNESS CONDITIONS HAVE
C              BEEN FORCED TO BE SATISFIED IN THE BRANCH
C              AND BOUND TREE.  A NEGATIVE VALUE MEANS
C              THE KUHN-TUCKER MULTIPLIER IS SET TO
C              ZERO, AND A POSITIVE VALUE MEANS A
C              CONSTRAINT IS FORCED TO BE TIGHT.
C      XBEST  THE ARRAY CONTAINING THE VALUES OF THOSE
C              VARIABLES WHICH ARE BASIC IN THE INCUMBENT
C              SOLUTION.
C      YCJ    THE ARRAY CONTAINING THE COEFFICIENTS OF
C              THE VARIABLES IN THE FOLLOWER'S OBJECTIVE
C              FUNCTION.
C      ZBEST  VALUE OF THE INCUMBENT SOLUTION OF THE
C              BLPP.
C      *****END DEFINITIONS*****
C
C      *****NOTE*****
C      X REFERS TO VARIABLES CONTROLLED BY THE LEADER;
C      Y REFERS TO VARIABLES CONTROLLED BY THE FOLLOWER;
C      U REFERS TO KUHN-TUCKER MULTIPLIERS ASSOCIATED
C          WITH CONSTRAINTS CONTAINING VARIABLES
C          CONTROLLED BY THE FOLLOWER AND WITH THE
C          NONNEGATIVE RESTRICTIONS ON THE VARIABLES
C          CONTROLLED BY THE FOLLOWER;
C      G REFERS TO THE SLACK OR SURPLUS VARIABLE
C          ASSOCIATED WITH EACH CONSTRAINT CONTAINING

```

```

C      VARIABLES CONTROLLED BY THE FOLLOWER.
C      *****END OF NOTE*****
C
C      ***SUMMARY OF INPUT FORMATS:***
C
C      CRITV, NUMX, NUMY, CONX, CONY, NUMITR              6I5
C
C      M, NSTRUC, MINMAX, ECHO, PRINT, BNDTYP            6I5
C
C      (I, ROWTYP(I), B(I))  I=1,M                      2I5,F10.0
C
C      XSUB, J, COLLEN, CJ, UJ, LJ                      A4, 2I5, 3F10.0
C
C      (I, A(I,J)) FOR I WITH A(I,J) NONZERO            I5,F10.0
C
C      INTEGER TOTCP1, VIRCP1, TOTCP2, VIRCP2, TOTCP, VIRCP2
C
C      ***DOUBLE PRECISION ARRAYS AND VARIABLES***
C      DOUBLE PRECISION B(100), BASCB(100), BASLB(100),
1          BASUB(100), BOUND, CANDA(100,6),
2          CANDCJ(6), CJ, COLA(100),
3          LJ, MEMR(11000), UJ, UZERO(100),
4          XZERO(100), YQ(100), Z
C
C      IF YOU HAVE ANY TWO-SIDED CONSTRAINTS (ROWTYP=2),
C      THEN DIMENSION BLOW(MAXM), THE SAME AS B(MAXM).
C      OTHERWISE, JUST DIMENSION BLOW(1).
C      NOTE: THIS VERSION OF XDEMO DOES NOT ALLOW
C            TWO-SIDED CONSTRAINTS.
C      DOUBLE PRECISION BLOW(1)
C
C      INTEGER ARRAYS AND VARIABLES.
C      INTEGER BNDTYP, COLLEN, COLMAX, DFEASQ, DTERM, DUNBR,
1          ERROR, FACTOR, IOERR, IOLOG, ITER,
2          ITER1, ITER2, LOOK, M, MAPI(9), MAPR(8),
3          MAXA, MAXM, MAXN, N, NTYPE2, PICK, PRINT,
4          TERMIN, UNBDDQ
C      INTEGER MEMI(11000)
C      INTEGER BASIS(100), CAND(6), CANDI(100,6), CANDL(6),
1          COLI(100), ROWTYP(100), STATUS(600)
C      DOUBLE PRECISION CJTEMP, ZTEMP, UTEMP
C      INTEGER ECHO, IOIN, ISEQ, JSEQ, JSEQ2, NSTRUC, MINMAX
C      CHARACTER*4 DUMY
C      CHARACTER*4 XSUB

```

C

```

C      VARIABLES NEEDED FOR BLPP BRANCH AND BOUND
      DOUBLE PRECISION BBVAR(600),XBEST(100),ZBEST,
1      BBTBIG
      INTEGER LEVEL,CRITV,W(600),BESTB(100),BESTAT(600),
1      L, IX,NUMX,NUMY,CONX,CONY,NSTOPV,NUMX1,
2      NUMXY,FLGUP,BLVAR,CONXY,OKBAS(100),
3      OKSTAT(600)
      DOUBLE PRECISION BBTEMP,BESTLB(600),BESTUB(600),
1      CJ1(600),UJO(600),YCJ(600)
      INTEGER DUMCNT,IVARU(600),VARTYP(600),
1      VCNT,FLAGUP(600)
C      *****COMMON VARIABLES
C      THE COMMON VARIABLES ARE USED AS NUMERICAL CHECKS
C      IN THE XMP SUBROUTINES.
      COMMON/XMPCOM/BIG,SMALL,ZL,ZLC,EPS1,EPS2,
1      EPS3,EPS4,EPS5,EPS6
      DOUBLE PRECISION BIG,SMALL,ZL,ZLC,EPS1,EPS2,
1      EPS3,EPS4,EPS5,EPS6
      COMMON/XMPLEN/LENI,LENMI,LENMR,LENR
      INTEGER LENI,LENMI,LENMR,LENR

C
C      XSUB IS USED AS A CHECK IN READING IN DATA.
      DATA XSUB/'XSUB'/

C
C
C
C
C      *****BODY OF PROGRAM*****

      IOIN=5
      IOERR=6
      IOLOG=6

C
C      ***PRINT HEADING***
C
      WRITE(IOLOG,917)
917  FORMAT(1H1///10X,'BLPP PROGRAM')
C      ***INITIALIZE XMP VARIABLES***
      MAXA=4000
      MAXM=100
      MAXN=600
      COLMAX=100

C
      PICK=6
      LOOK=200
      FACTOR=50

```

```

C
    LENI=11000
    LENR=11000
    LENMI=9
    LENMR=8
C
C    ***READ IN PROBLEM INFORMATION***
    READ(IOIN,900) CRITV,NUMX,NUMY,CONX,CONY,NUMITR
C
    READ(IOIN,900) M,NSTRUC,MINMAX,ECHO,PRINT,BNDTYP
900  FORMAT(6I5)
    WRITE (6,898) NUMX,NUMY,CONX,CONY
898  FORMAT(' NUMX ',I4,' NUMY ',I4,' CONX ',I4,
1      ' CONY ',I4)
C
C    ***INITIALIZE BRANCH AND BOUND VARIABLES***
    NUMXY=NUMX+NUMY
    NUMXY1=NUMXY+1
    NUMX1=NUMX+1
    CONXY=CONX+CONY
    NSTOPV=CRITV+NUMX
    BLVAR=NUMX+NUMY+CONY+NUMY+CONY
    ZBEST=-1.0D25
    NODE=0
    NODBST=0
    LEVEL=0
    DO 1199 J=1,NSTRUC
1199  BBVAR(J)=0.0
    DO 1198 J=1,NSTOPV
1198  W(J)=0
C
C
    IF(MINMAX .EQ. 1)WRITE(IOLOG,960)
    IF(MINMAX .EQ. -1)WRITE(IOLOG,961)
960  FORMAT(1H0,10X,26HTHIS IS A MAXIMIZATION RUN//)
961  FORMAT(1H0,10X,26HTHIS IS A MINIMIZATION RUN//)
C
C
    WRITE(IOLOG,918)M
918  FORMAT(1H0,5X,I6,2X,11HCONSTRAINTS)
    WRITE(IOLOG,919)NSTRUC
919  FORMAT(1H0,5X,I6,2X,20HSTRUCTURAL VARIABLES///)
C
C

```



```

      CALL XMAPS(BNDTYP, IOERR, MAPI, MAPR, MAXA, MAXM
1      MAXN, MEMI, MEMR)

C
C
C      READ IN THE ROW TYPES AND THE RIGHT-HAND-SIDE.
C      ISEQ IS THE ROW NUMBER, FOR THE USER'S
C      CONVENIENCE.
C
      NOSX=NUMX
      DO 100 I=1,M
      READ(IOIN,902) ISEQ,ROWTYP(I),B(I)
      IF (I.LE.CONXY)GO TO 100
      NOSX=NOSX+1
      YCJ(NOSX)=-B(I)
100 CONTINUE
902 FORMAT(2I5,F13.3)

C
C
      IF(ECHO .LT. 1)GO TO 110
      WRITE(IOLOG,913)
913 FORMAT(///10X,7THROW NO.,10X,8THROW TYPE,10X,
X 16H RIGHT-HAND-SIDE/)
      DO 105 I=1,M
105 WRITE(IOLOG,914) I,ROWTYP(I),B(I)
914 FORMAT(10X,I7,10X,I8,10X,D16.8)
110 CONTINUE

C
C
C      N IS THE CURRENT NUMBER OF VARIABLES.
C      N IS INCREMENTED BY XADDAJ AS EACH COLUMN
C      IS ADDED.
C      N=0
C      JSEQ2=0

C
C
C      ***READ IN THE COLUMNS ONE AT A TIME***
C
C      JSEQ IS A SEQUENCE NUMBER FOR THE USER'S
C      CONVENIENCE
C      (XMP NUMBERS THE COLUMNS SEQUENTIALLY AS THEY ARE
C      SENT TO XADDAJ AND RETURNS THIS NUMBER IN IDNO)
C
      INTCNT=0
C
      DO 120 J=1,NSTRUC

```

```

      READ(IOIN,904) DUMY,JSEQ,COLLEN,CJ,
1          UJ,LJ,VARTYP(J)
904      FORMAT(A4,2I5,F15.5,2F10.0,I2)
          UJO(J)=UJ
          CJ1(J)=CJ
          IF(DUMY .EQ. XSUB)GO TO 1105
          WRITE(IOERR,901)JSEQ2
901      FORMAT(31H ERROR IN INPUT FILE AFTER XSUB,I5)
          STOP
1105     JSEQ2=JSEQ
C      FLIP THE SIGN OF THE OBJECTIVE FUNCTION
C      COEFFICIENT IF WE ARE MINIMIZING.
          CJTEMP=CJ
          IF(MINMAX .EQ. -1) CJTEMP=-CJ
          DO 130 K=1,COLLEN
              READ(IOIN,905) COLI(K),COLA(K)
130      CONTINUE
905      FORMAT(I5,F15.5)
C
          CALL XADDAJ(CJTEMP,COLA,COLI,COLLEN,COLMAX,
1              IOERR,IDNO,MAPI,MAPR,MEMI,MEMR,N)
C
          IF(BNDTYP .GT. 1)GO TO 112
          LJ=0.0
          UJ=BIG
          GO TO 115
112      IF(BNDTYP .GT. 2)GO TO 113
          ERROR=3
          CALL XSTOP(ERROR,IOERR)
113      IF(BNDTYP .EQ. 3) LJ=0.0
C
          CALL XADDUB(BNDTYP,IOERR,IDNO,LJ,MAPI,MAPR,
1              MEMI,MEMR,UJ)
C
115      IF(ECHO .LT. 1)GO TO 120
          WRITE(IOLOG,915)IDNO,CJ,UJ,LJ
915      FORMAT(/1X,2HJ=,I5,5X,3HCJ=,D16.8,5X,
1          3HUJ=,D16.8,5X,3HLJ=,D16.8)
          DO 118 K=1,COLLEN
              WRITE(IOLOG,916) COLI(K),COLA(K)
118      CONTINUE
916      FORMAT(13X,I6,5X,D16.8)
120 CONTINUE
C
C      ***START ALL OF THE STRUCTURAL VARIABLES AT THEIR

```

```

C          LOWER BOUNDS***
C
      DO 170 J=1,N
        STATUS(J)=0
170 CONTINUE
C
      CALL XSLACK(B,BASCB,BASIS,BASLB,BASUB,
1          BLOW,BNDTYP,BOUND,
2          COLA,COLI,COLMAX,IOERR,
3          M,MAPI,MAPR,MAXM,MAXN,MEMI,
4          MEMR,N,ROWTYP,STATUS,
5          UZERO,XBZERO,Z)
C
C      CPTIME IS A UTILITY USED ON THE UNIVERSITY
C      OF TEXAS IBM COMPUTER TO FIND CPU TIME
C      CALL CPTIME(TOTCP1,VIRCP1)
C
      GO TO 4500
C
C      ***BEGIN BLPP BRANCH AND BOUND***
1999 CONTINUE
      IF (NODE.LT.NUMITR) GO TO 4005
      FLGOUT=1.0
      GO TO 6000
4005 IF (TERMIN.NE.1) GO TO 2500
      DO 4010 K=1,M
        OKBAS(K)=BASIS(K)
4010 CONTINUE
      DO 4020 K=1,N
        OKSTAT(K)=STATUS(K)
4020 CONTINUE
      IF (ZTEMP.LE.ZBEST) GO TO 3100
C      ***DETERMINE VALUE OF VARIABLES IN SOLUTION AT
C      CURRENT NODE***
      DO 670 I=NUMX1,NSTRUC
        IF (STATUS(I)) 400,500,600
400      IF (STATUS(I).EQ.-2 .OR. STATUS(I).EQ.-3)
1          GO TO 430
        IF (STATUS(I).EQ.-4) GO TO 500
        CALL XGETUB(BNDTYP,IOERR,I,LJ,MAPI,MAPR,
1          MEMI,MEMR,UJ)
        BBVAR(I)=UJ
        GO TO 670
430      DO 440 J=1,M
          IF(BASIS(J).NE.I) GO TO 440

```

```

        BBVAR(I)=XBZERO(J)
        GO TO 670
440    CONTINUE
500    IF (BNDTYP.NE.4) GO TO 555
        CALL XGETUB(BNDTYP,IOERR,I,LJ,MAPI,MAPR,
1         MEMI,MEMR,UJ)
        IF (LJ.EQ.0.0) GO TO 555
        BBVAR(I)=LJ
        GO TO 670
555    BBVAR(I)=0.0
        GO TO 670
600    IX=STATUS(I)
        BBVAR(I)=XBZERO(IX)
670    CONTINUE
C     ***FEASIBILITY CHECK/SELECT BRANCHING VARIABLE***
C     THIS SECTION CHECKS TO SEE IF EACH COMPLEMENTARY
C     SLACKNESS CONDITION IS SATISFIED (IE IF U*G=0).
C     IF YES, THEN FEASIBLE SOLUTION FOUND. IF NO,
C     THEN PICK LARGEST VALUE OF U*G AND BRANCH ON
C     U (IE MAKE U ZERO)
C     FLAG=1 MEANS WORKING WITH A VARIABLE AT
C           ITS LOWER BOUND.
C     FLAG=2 MEANS WORKING WITH A VARIABLE AT
C           ITS UPPER BOUND.
C     BBTBIG=0.0
C     VCNT=0
C     FLGUP=1
        DO 1200 I=NUMX1,NSTOPV
            IF (VARTYP(I).EQ.0 .OR. I.GT.NUMXY) GO TO 1210
            VCNT=VCNT+1
            BBTEMP=(UJO(I)-BBVAR(I))*BBVAR(BLVAR+VCNT)
            IF (BBTEMP.LE.BBTBIG) GO TO 1210
            BBTBIG=BBTEMP
            KK=I
            DUMCNT=VCNT
            FLGUP=2
1210    BBTEMP=BBVAR(I)*BBVAR(I+CRITV)
            IF (BBTEMP.LE.BBTBIG) GO TO 1200
            BBTBIG=BBTEMP
            KK=I
            FLGUP=1
1200    CONTINUE
        IF (BBTBIG.LE.0.000001) GO TO 2000
C     ***BRANCH ON SELECTED VARIABLE***
C     BRANCHES ON SELECTED U VARIABLE(IE SETS U TO 0)

```

```

C      W-ARRAY USED TO TELL WHICH VARIABLE OF A U-G PAIR
C      IS SET.  W(I)=K MEANS VARIABLE K SET AT LEVEL I OF
C      TREE.  IF K<0 THAT MEANS ALTERNATIVE OF U-G PAIR
C      NOT YET CONSIDERED.
1500  LEVEL=LEVEL+1
      NODE=NODE+1
      FLAGUP(LEVEL)=FLGUP
      GO TO (1510,1520) FLAGUP(LEVEL)
1510  W(LEVEL)=-KK
      L=KK+CRITV
      UJ=0.0
      LJ=0.0
      CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
      GO TO 4000
1520  W(LEVEL)=-KK
      L=BLVAR+DUMCNT
      UJ=0.0
      LJ=0.0
      IVARU(LEVEL)=DUMCNT
      CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
      GO TO 4000
C      ***A NEW INCUMBENT SOLUTION FOUND***
2000  DO 2100 K=1,M
      XBEST(K)=XBZERO(K)
      BESTB(K)=BASIS(K)
2100  CONTINUE
      DO 2200 K=1,N
      BESTAT(K)=STATUS(K)
2200  CONTINUE
      DO 2250 K=1,NSTRUC
      CALL XGETUB(BNDTYP,IOERR,J,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
      BESTLB(K)=LJ
      BESTUB(K)=UJ
2250  CONTINUE
      NODBST=NODE
      ZBEST=ZTEMP
      IF (LEVEL.EQ.0) GO TO 6000
      GO TO 3100
2500  IF (TERMIN.EQ.2) GO TO 5000
      IF (TERMIN.NE.1 .AND. NODE.EQ.0) GO TO 5000
C      ***BACKTRACK***
C      FINDS A LIVE NODE, FREES U VAR, AND

```

```

C      SETS G VAR TO 0
C      ALSO FREES VARIABLES WHICH HAVE HAD BOTH
C      ALTERNATIVES CONSIDERED.
3100  CONTINUE
      IF (LEVEL.EQ.0) GO TO 6000
      IF(W(LEVEL).LT.0) GO TO 3200
      L=W(LEVEL)
      UJ=UJO(L)
      LJ=0.0
      CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
      W(LEVEL)=0
      LEVEL=LEVEL-1
      GO TO 3100
C      ***RANCH TO LIVE NODE***
3200  NODE=NODE+1
      GO TO (3210,3220) FLAGUP(LEVEL)
C      ***WORK WITH VARIABLES WHICH ARE UNBOUNDED***
C
C      FREE U
3210  L=-W(LEVEL)+CRITV
      UJ=UJO(L)
      LJ=0.0
      CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
C      SET A G TO ZERO
      W(LEVEL)=-W(LEVEL)
      L=W(LEVEL)
      UJ=0.0
      LJ=0.0
      CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
      GO TO 4000
C      ***WORK WITH VARIABLES WHICH ARE BOUNDED***
3220  L=BLVAR+IVARU(LEVEL)
C      FREE U
      UJ=UJO(L)
      LJ=0.0
      CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
C      SET G TO ITS UPPER BOUND
      W(LEVEL)=-W(LEVEL)
      L=W(LEVEL)
      UJ=UJO(L)
      LJ=UJO(L)

```

```

3236 CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
                MEMI,MEMR,UJ)
C      ***SECTION SETS BASIS AND STATUS ARRAYS USED BY
C      XMP SUBROUTINE XSTART WHICH CAN START THE
C      PRIMAL SIMPLEX METHOD FROM ANY GIVEN BASIS***
4000 DO 4001 K=1,M
      BASIS(K)=OKBAS(K)
4001 CONTINUE
      DO 4002 K=1,N
      STATUS(K)=OKSTAT(K)
4002 CONTINUE
C      ***CALLS XMP LINEAR PROGRAMMING SUBROUTINES***
      CALL XSTART(B,BASCB,BASIS,BASLB,BASUB,
1          BNDTYP,BOUND,
2          COLA,COLI,COLMAX,IOERR,
3          M,MAPI,MAPR,MAXM,MAXN,MEMI,MEMR,
4          N,NTYPE2,SCODE,STATUS,
5          UZERO,XBZERO,Z)
C
4500 CALL XPRIML(B,BASCB,BASIS,BASLB,BASUB,BNDTYP,
1          BOUND,COLA,COLI,COLMAX,
2          FACTOR,IOERR,IOLOG,ITER1,ITER2,
3          LOOK,M,MAPI,MAPR,MAXM,MAXN,MEMI,MEMR,
4          N,NTYPE2,PICK,PRINT,STATUS,TERMIN,
5          UNBDDQ,UZERO,XBZERO,YQ,Z)
C
C      FLIP THE SIGN OF THE OBJECTIVE FUNCTION VALUE IF
C      PROBLEM IS MINIMIZATION.
C      ZTEMP=Z
C      IF(MINMAX .EQ. -1) ZTEMP=-ZTEMP
C
C
C      GO TO 1999
C
C      PROBLEM DOES NOT HAVE AN OPTIMAL SOLUTION
C
5000 WRITE(IOLOG,7000) TERMIN
7000 FORMAT(1H1///,10X,'THE PROBLEM IS INFEASIBLE ',
1          'OR UNBOUNDED',I3)
      GO TO 314
C
C      ***PRINT OUT THE SOLUTION***
C
C      CPTIME IS A UTILITY USED ON THE UNIVERSITY
C      OF TEXAS IBM COMPUTER TO FIND CPU TIME

```

```

6000 CALL CPTIME(TOTCP2,VIRCP2)
      TOTCP=TOTCP2-TOTCP1
      VIRCP=VIRCP2-VIRCP1
      IF (FLGOUT.EQ.1.0) WRITE (6,1985)
1985 FORMAT('THE ABOVE SOLUTION IS NOT OPTIMAL.  ',
1       'THE BRANCH AND BOUND TREE HAS ',
2       'REACHED ITS MAX SIZE')
      CALL XPRINT(BESTB,BNDTYP,BOUND,
1       IOERR,IOLOG,NUMXY,M,
2       MAPI,MAPR,MAXM,MAXN,MEMI,MEMR,
3       N,NTYPE2,BESTAT,XBEST,ZBEST,
4       UJO,BESTLB,BESTUB)
C
C
      IF (FLGOUT.EQ.1.0) WRITE (6,1984)
1984 FORMAT('THE ABOVE SOLUTION IS NOT OPTIMAL.  ',
1       'THE BRANCH AND BOUND TREE HAS ',
2       'REACHED ITS MAX SIZE')
      WRITE (6,1988) NODE,NODBST
1988 FORMAT(' NUMBER OF NODES ',I6,' BEST NODE ',I6)
C
314 CONTINUE
C
      WRITE (6,70) TOTCP,VIRCP
70  FORMAT(3X,'TOTAL CPTIME =',I6,3X,
1       'VIRTUAL CPTIME =',I6)
      STOP
      END
C
C
      SUBROUTINE XPRINT(BASIS,BNDTYP,BOUND,
1       IOERR,IOLOG,NUMXY,M,
2       MAPI,MAPR,MAXM,MAXN,MEMI,
3       MEMR,N,NTYPE2,STATUS,XBZERO,
4       Z,UJO,BESTLB,BESTUB)
C
C
      MODIFIED XMP SUBROUTINE.  PRINTS OUT VALUES OF
      OF VARIABLES.
C
      INTEGER MAXM,MAXN
      DOUBLE PRECISION BOUND,MEMR(LENR),XBZERO(MAXM),Z
      INTEGER BNDTYP,IOERR,IOLOG,M,N,NTYPE2
      INTEGER MAPI(LENMI),MAPR(LENMR)
      INTEGER BASIS(MAXM),MEMI(LENI),STATUS(MAXN)
C      *****LOCAL VARIABLES*

```



```

      DOUBLE PRECISION LJ,UJ,VALUE,BESTLB(600),
1      BESTUB(600)
      INTEGER IX,J,NUMXY
C      *****COMMON VARIABLES
      COMMON/XMPLEN/LENI,LENMI,LENMR,LENR
      INTEGER LENI,LENMI,LENMR,LENR
C
      WRITE(IOLOG,901)
901  FORMAT(1H0,28H  XPRINT...CURRENT SOLUTION)
      WRITE(IOLOG,902)M,N
902  FORMAT(1H0,I6,20H LINEAR CONSTRAINTS,,5X,I6,
1      10H VARIABLES)
      WRITE(IOLOG,903)
903  FORMAT(1H0,25H  AN SB FOLLOWING STATUS ,
1      32HINDICATES A SUPER-BASIC VARIABLE)
C
      WRITE(IOLOG,904)
904  FORMAT(1H0,10H  VARIABLE,5X,10H  STATUS,5X,
1      15H  VALUE)
C
C      PRINT OUT THE VALUES OF ALL OF THE BASIC
C      VARIABLES AND ALL OF THE NON-BASIC VARIABLES
C      THAT ARE AT NON-ZERO BOUNDS.
      DO 280 J=1,NUMXY
          IF(STATUS(J))200,250,260
200      IF(STATUS(J).EQ.-2 .OR. STATUS(J).EQ.-3)
1          GO TO 230
          IF(STATUS(J).EQ.-4)GO TO 250
C      HERE FOR NON-BASIC VARIABLES AT THEIR
C      UPPER BOUNDS.
          IF(BNDTYP.LT.3)GO TO 210
          VALUE=BESTUB(J)
          GO TO 270
210      IF(BNDTYP.LT.2)GO TO 220
          IF(J.GT.NTYPE2)GO TO 220
          VALUE=BOUND
          GO TO 270
220      WRITE(IOLOG,905) J
905      FORMAT(16H ERROR: VARIABLE,I6,
1          16H IS AT ITS UPPER,
2          20H BOUND OF +INFINITY.)
          GO TO 280
C
C      HERE FOR FREE AND ARTIFICIAL VARIABLES.
230      DO 240 I=1,M

```

```

        IF(BASIS(I) .NE. J)GO TO 240
        VALUE=XBZERO(I)
        GO TO 270
240     CONTINUE
        GO TO 280
C      HERE FOR NON-BASIC VARIABLES AT THEIR
C      LOWER BOUNDS.
250     IF(BNDTYP .NE. 4)GO TO 280
        VALUE=BESTLB(J)
        GO TO 270
C
C      HERE FOR THE BASIC VARIABLES (EXCEPT FREE
C      AND ARTIFICIALS)
260     IX=STATUS(J)
        VALUE=XBZERO(IX)
        IF(IX .LE. M)GO TO 270
C
C      HERE FOR SUPER-BASIC VARIABLES.
        WRITE(IOLOG,908)J,STATUS(J),VALUE
908     FORMAT(1X,I10,5X,I10,1X,2HSB,2X,D15.8)
        GO TO 280
C
270     WRITE(IOLOG,906) J,STATUS(J),VALUE
906     FORMAT(1X,I10,5X,I10,5X,D25.12)
C
280 CONTINUE
C
C      PRINT THE OBJECTIVE VALUE.
        WRITE(IOLOG,907) Z
907 FORMAT('  VALUE OF THE LINEAR OBJECTIVE ',
1         'FUNCTION = ',D15.8)
C
C      RETURN
        END

```

A.6 Codes for the Mixed Integer BLPP

The codes for the mixed integer BLPP include a random problem generator, a program which can be used to input a mixed integer BLPP, and the algorithms for the mixed integer BLPP. The listing of the program which can be used to input a mixed integer BLPP is presented in section A.5.2.

A.6.1 The Mixed Integer BLPP Random Problem Generator

The listing below presents the code which generates a random mixed integer BLPP. The program calls the random number generator URAND which is presented in section A.2. The program requires nine inputs. They are

- the total number of variables controlled by the leader;
- the total number of variables controlled by the follower;
- the number of integer variables controlled by the leader;
- the number of integer variables controlled by the follower;
- the number of constraints with no variables controlled by the follower;

the number of constraints with variables controlled
by the follower;

the maximum number of vertices allowed in the
branch and bound tree;

the maximum number of nonzero elements in a column
of the A or B matrix;

the random number seed.

The program is listed below.

PROGRAM MIXGEN

*****PURPOSE

PROGRAM GENERATES RANDOM LINEAR MIXED INTEGER
 BILEVEL PROGRAMMING PROBLEMS (BLPP). THE FUNCTION
 URAND PROVIDES RANDOM NUMBERS. REFER TO THE
 SECTION ON THE FUNCTION URAND FOR MORE
 INFORMATION AND A LISTING OF THE FUNCTION.

VARIABLE DEFINITIONS

ACC ARRAY WHICH CONTAINS THE COEFFICIENTS OF
 EACH VARIABLE IN THE A- AND B-MATRICES OF
 THE BLPP.

B THE RIGHT HAND SIDE ARRAY.

BNDTYP 1 MEANS LOWER BOUND=0, UPPER
 BOUND=+INFINITY FOR EVERY NON-FREE
 VARIABLE;
 2 MEANS LOWER BOUND=0, UPPER BOUND=BOUND
 FOR EVERY NON-FREE STRUCTURAL VARIABLE
 3 MEANS LOWER BOUND=0 FOR EVERY NON-FREE
 VARIABLE;
 4 MEANS BOTH BOUNDS ARE GENERAL.

CJ1 ARRAY CONTAINING THE LEADER'S OBJECTIVE
 FUNCTION COEFFICIENTS.

CJ2 ARRAY CONTAINING THE FOLLOWER'S OBJECTIVE
 FUNCTION COEFFICIENTS.

COLA IS USED TO HOLD THE NON-ZERO COEFFICIENTS
 OF A MATRIX COLUMN.

COLI IS USED TO HOLD THE ROW NUMBERS
 CORRESPONDING TO THE NON-ZEROS OF A MATRIX
 COLUMN.

COLLEN THE NUMBER OF NONZEROS IN EACH COLUMN OF
 THE A- AND B-MATRICES OF THE BLPP.

CONX NUMBER OF CONSTRAINTS CONTAINING NO
 VARIABLES CONTROLLED BY THE FOLLOWER.

CONY NUMBER OF CONSTRAINTS CONTAINING VARIABLES
 CONTROLLED BY THE FOLLOWER.

CRITV NUMBER OF COMPLEMENTARY SLACKNESS
 CONDITIONS WHICH MUST BE SATISFIED IN
 SOLVING THE BLPP IN ITS EQUIVALENT KUHN-
 TUCKER REPRESENTATION.

IY RANDOM NUMBER SEED OF IY.

LJ IS USED TO HOLD THE LOWER BOUND ON A
 SINGLE VARIABLE.

```

C      M      NUMBER OF CONSTRAINTS IN THE EQUIVALENT
C      KUHN-TUCKER REPRESENTATION OF THE BLPP.
C      MAXCOL  MAXIMUM NUMBER OF NONZEROS IN ANY ONE
C      COLUMN OF THE BLPP.
C      MINMAX  FLAG INDICATING NATURE OF THE PROBLEM.
C      1 MEANS MAXIMIZATION; -1 MEANS
C      MINIMIZATION. FIXED AT 1 IN THIS PROGRAM.
C      NSTRUC  NUMBER OF VARIABLES IN THE KUHN-TUCKER
C      REPRESENTATION OF THE MIXED INTEGER BLPP.
C      NUMCON  THE NUMBER OF CONSTRAINTS IN THE BLPP.
C      NUMITR  THE MAXIMUM NUMBER OF VERTICES IN THE
C      BRANCH AND BOUND TREE CREATED WHEN SOLVING
C      THE BLPP.
C      NUMX    THE NUMBER OF VARIABLES CONTROLLED BY THE
C      LEADER.
C      NUMXY   THE NUMBER OF VARIABLES IN THE BLPP.
C      NUMY    THE NUMBER OF VARIABLES CONTROLLED BY THE
C      FOLLOWER.
C      PRINT   AN XMP VARIABLE. SETS THE AMOUNT OF
C      OUTPUT CREATED BY XMP. SEE THE XMP
C      DICTIONARY FOR FURTHER INFORMATION.
C      ROWTYP  ARRAY WHICH INDICATES THE TYPE OF
C      CONSTRAINT:
C      +1 MEANS LESS THAN OR EQUAL TO;
C      0 MEANS EQUATION;
C      -1 MEANS GREATER THAN OR EQUAL TO.
C      UJ      IS USED TO HOLD THE UPPER BOUND ON A
C      SINGLE VARIABLE.
C      UROW    ARRAY CONTAINS VALUES USED TO CREATE THE
C      KUHN-TUCKER STATIONARITY CONDITIONS OF THE
C      FOLLOWER'S PROBLEM.
C      VARTYP  INDICATES VARIABLE TYPE:
C      0 MEANS A VARIABLE IS REAL VALUED;
C      1 MEANS A VARIABLE IS INTEGER;
C      2 MEANS A VARIABLE IS BOUNDED AND INTEGER;
C      3 MEANS A VARIABLE IS BOUNDED AND REAL
C      VALUED.
C      XINT    NUMBER OF INTEGER VARIABLES CONTROLLED BY
C      THE LEADER.
C      YINT    NUMBER OF INTEGER VARIABLES CONTROLLED BY
C      THE FOLLOWER.
C      ***END VARIABLE DEFINITIONS***
C
C      ***NOTE***
C      X REFERS TO VARIABLES CONTROLLED BY THE LEADER;

```

```

C      Y REFERS TO VARIABLES CONTROLLED BY THE FOLLOWER;
C      U REFERS TO KUHN-TUCKER MULTIPLIERS ASSOCIATED
C      WITH CONSTRAINTS CONTAINING VARIABLES
C      CONTROLLED BY THE FOLLOWER AND WITH THE
C      NONNEGATIVE RESTRICTIONS ON THE VARIABLES
C      CONTROLLED BY THE FOLLOWER;
C      G REFERS TO THE SLACK OR SURPLUS VARIABLE
C      ASSOCIATED WITH EACH CONSTRAINT CONTAINING
C      VARIABLES CONTROLLED BY THE FOLLOWER.
C      ***END OF NOTE***
C
      REAL UROW(200,200),COLA(200),B(200),ACC(200,200),
1      CJ1(200),CJ2(200),UJ,LJ
      INTEGER CNT,COLI(200),COLLEN(200),CONY,CONX,CONXY,
1      CRITV,PRINT,ECHO,BNDTYP,ROWTYP(200),
2      VARTYP(200),XINT,YINT
      CHARACTER*4 DUMY
      AONE=1.0
      IONE=1
      ZERO=0.0
      IZERO=0
      DUMY='XSUB'
C      ***READ IN PROBLEM INFORMATION***
      READ (5,*) NUMX,NUMY,XINT,YINT,CONX,CONY,MAXNDS,
1      MAXCOL,IY
C
      MINMAX=1
      ECHO=0
      PRINT=0
      BNDTYP=4
      UJ=200000.0
      LJ=0.0
      CRITV=NUMY+CONY
      M=NUMY+CONY+CONX
      NUMXY=NUMX+NUMY
      NUMXY1=NUMXY+1
      NSTRUC=NUMY+NUMY+CONY+CONY+NUMX
      CONX1=CONX+1
      NUMX1=NUMX+1
      CONXY=CONX+CONY
      IF (CONX.EQ.0) GO TO 105
C      ***COMPUTE RHS OF X CONSTRAINTS***
      DO 100 I=1,CONX
          B(I)=URAND(IY)*50.0
100  CONTINUE

```

```

C      ***COMPUTE RHS OF CONSTRAINTS WITH VARIABLES
C      CONTROLLED BY THE FOLLOWER***
105 DO 110 I=CONX1,CONXY
      B(I)=URAND(IY)*50.0
110 CONTINUE
C      ***DETERMINE SIGN OF CONSTRAINTS
C      0.7 < AND 0.3 >***
      DO 115 I=1,CONXY
        DUM=URAND(IY)
        ROWTYP(I)=1
        IF (DUM.LE.0.3) ROWTYP(I)=-1
115 CONTINUE
C      ***COMPUTE COEFFICIENTS OF LEADER AND FOLLOWER
C      OBJECTIVE FUNCTIONS***
      DO 120 I=1,NUMXY
        CJ1(I)=URAND(IY)*50.0-25.0
        CJ2(I)=URAND(IY)*50.0-25.0
120 CONTINUE
      MAXC2=MAXCOL+MAXCOL
C      GENERATE THE A- B-MATRICES OF THE BLPP***
      DO 130 I=1,NUMXY
        IROW=0
        CNT=0
        IF (I.GE.NUMX1) IROW=CONX
        DO 140 J=1,MAXC2,2
          YA=URAND(IY)*61.0-15.0
          ROWNUM=URAND(IY)*3.0+1.0
          IROW=ROWNUM+IROW
C      ***X & Y COEFFICIENTS ALLOWED IN Y CONST.
C      NO Y'S ALLOWED IN X CONST***
          IF(IROW.GT.CONXY) GO TO 145
          ACC(I,J)=IROW
          ACC(I,J+1)=YA
          UROW(I,IROW)=YA
          CNT=CNT+1
140 CONTINUE
145 COLLEN(I)=CNT
130 CONTINUE
C      ***RANDOMLY DETERMINE WHICH X & Y VARIABLES
C      ARE INTEGER***
      INTCNT=0
      PERCNT=FLOAT(XINT)/FLOAT(NUMX)
310 DO 300 I=1,NUMX
      IF (INTCNT.GE.XINT) GO TO 305
      IF (VARTYP(I).EQ.1) GO TO 300

```



```

      VV=URAND(IY)
      IF (VV.GT.PERCNT) GO TO 300
      VARTYP(I)=1
      INTCNT=INTCNT+1
300  CONTINUE
      IF (INTCNT.LT.XINT) GO TO 310
305  INTCNT=0
      PERCNT=FLOAT(YINT)/FLOAT(NUMY)
315  DO 320 I=NUMX1,NUMXY
      IF (INTCNT.GE.YINT) GO TO 325
      IF (VARTYP(I).EQ.1) GO TO 320
      VV=URAND(IY)
      IF (VV.GT.PERCNT) GO TO 320
      VARTYP(I)=1
      NSTRUC=NSTRUC+1
      INTCNT=INTCNT+1
320  CONTINUE
      IF (INTCNT.LT.YINT) GO TO 315
C    ***WRITE PROBLEM INFORMATION***
325  WRITE (6,905) CRITV,NUMX,NUMY,CONX,CONY,XINT,
      1      YINT,MAXNDS
905  FORMAT (8I5)
      WRITE (6,908) M,NSTRUC,MINMAX,ECHO,PRINT,BNDTYP
908  FORMAT(6I5)
C    ***WRITE CONSTRAINTS OF THE BLPP***
      DO 150 I=1,CONX
      WRITE (6,910) I,ROWTYP(I),B(I)
150  CONTINUE
      DO 155 I=CONX1,CONXY
      WRITE (6,910) I,IZERO,B(I)
155  CONTINUE
910  FORMAT(2I5,F13.3)
C    ***WRITE RHS OF U-ROWS***
      DO 160 I=1,NUMY
      K=I+CONXY
      J=I+NUMX
      WRITE (6,910) K,IZERO,-CJ2(J)
160  CONTINUE
C    ***WRITE X VARIABLE INFO AND COLUMNS***
      DO 230 I=1,NUMX
      ICOLL=COLLEN(I)
      WRITE (6,915) DUMY,I,ICOLL,CJ1(I),UJ,LJ,
      1      VARTYP(I)
      ICOLL2=2*ICOLL
      DO 235 J=1,ICOLL2,2

```

```

        IB=ACC(I,J)
234      WRITE (6,920) IB,ACC(I,J+1)
235      CONTINUE
230 CONTINUE
915  FORMAT(A4,2I5,F15.5,2F10.0,I2)
920  FORMAT(I5,F15.5)
C    ***WRITE Y VARIABLE INFO AND COLUMNS***
      DO 170 I=NUMX1,NUMXY
        ICOLL=COLLEN(I)
        ICOLL2=2*ICOLL
        WRITE (6,915) DUMY,I,ICOLL,CJ1(I),UJ,LJ,
1          VARTYP(I)
        DO 180 J=1,ICOLL2,2
          IB=ACC(I,J)
          WRITE (6,920) IB,ACC(I,J+1)
180    CONTINUE
170 CONTINUE
C    WRITE G'S OF Y CONSTRAINTS***
      DO 190 I=1,CONY
        ISEQ=NUMXY+I
        WRITE (6,915) DUMY,ISEQ,IONE,ZERO,UJ,LJ,IZERO
        IDUM=I+CONX
        AONE=AONE*ROWTYP(IDUM)
        WRITE (6,920) IDUM,AONE
        AONE=1.0
190 CONTINUE
C    ***WRITE THE U'S OF THE Y .GE. 0 CONSTRAINTS***
      DO 200 I=1,NUMY
        ISEQ=NUMXY+CONY+I
        IROW=I+CONXY
        WRITE (6,915) DUMY,ISEQ,IONE,ZERO,UJ,LJ,IZERO
        WRITE (6,920) IROW,AONE
200 CONTINUE
C    ***WRITE THE U'S WHICH ARE KUHN-TUCKER
C    MULTIPLIERS Y CONSTRAINTS DERIVED FROM
C    EQUATION
C      GRAD(F(Y))-U*GRAD(G(Y))=0***
      DO 210 I=CONX1,CONXY
        CNT=0
        DO 215 J=NUMX1,NUMXY
          IF(UROW(J,I).EQ.0.0) GO TO 215
          CNT=CNT+1
          COLA(CNT)=-UROW(J,I)
          COLI(CNT)=J+CONXY-NUMX
215    CONTINUE

```

```
ISEQ=I+CONY+NUMY+NUMXY-CONX
WRITE (6,915) DUMY,ISEQ,CNT,ZERO,UJ,ZERO,IZERO
DO 220 K=1,CNT
    WRITE (6,920) COLI(K),COLA(K)
220  CONTINUE
210  CONTINUE
C    ***WRITE U'S WHICH ARE FOR THE INTEGER Y'S
C    POSSIBLE UPPER BOUND***
ISEQ=CONY+CONY+NUMY+NUMY+NUMX
CNT=0
DO 400 I=NUMX1,NUMXY
    CNT=CNT+1
    IF (VARTYP(I).EQ.0) GO TO 400
    ISEQ=ISEQ+1
    WRITE (6,915) DUMY,ISEQ,IONE,ZERO,UJ,ZERO,IZERO
    IROW=CONXY+CNT
    WRITE (6,920) IROW,-AONE
400  CONTINUE
STOP
END
```

A.6.2 The Algorithms for the Mixed Integer BLPP

The following program is the computerized version of Algorithm 1 presented in Chapter 6. The program is documented to show where modifications are made to create Algorithms 2 through 8. The XMP variables are defined in section A.4. The variables local to the main program are defined in the code. In line documentation is also included. The program is listed below.

PROGRAM BLPPM1

*****PURPOSE:

THIS IS THE PROGRAM WHICH READS IN THE MIXED INTEGER BILEVEL PROGRAMMING PROBLEM AND SOLVES IT USING A BRANCH AND BOUND PROCEDURE. THIS PROGRAM USES THE SUBROUTINE LIBRARY XMP TO SOLVE THE SUBPROBLEMS WHICH ARE CREATED IN THE BRANCH AND BOUND TREE. XMP IS DESIGNED TO SOLVE LINEAR PROGRAMMING PROBLEMS USING THE PRIMAL OR DUAL SIMPLEX METHOD.

THIS PROGRAM READS THE PROBLEM DATA INTO XMP COMPATIBLE ARRAYS. IT THEN SOLVES THE LINEAR MIXED INTEGER BILEVEL PROGRAMMING PROBLEM USING THE XMP SUBROUTINE LIBRARY. EACH SUBPROBLEM IS SOLVED USING THE PRIMAL SIMPLEX METHOD.

THIS PROGRAM IS A GENERAL PURPOSE FRONT END FOR READING IN AND SOLVING A LINEAR MIXED INTEGER BILEVEL PROGRAMMING PROBLEM. IT DOES NOT TAKE ANY ADVANTAGE OF SPECIAL PROBLEM TYPE OR STRUCTURE. FOR MOST SERIOUS APPLICATIONS YOU WILL WANT TO WRITE A MATRIX GENERATOR THAT READS IN DATA TABLES AND GENERATES THE LP COEFFICIENT MATRIX ONE COLUMN AT A TIME. THE FORTRAN PROGRAM BLPPREAD CAN BE USED TO READ IN AND CONVERT THE DATA OF A MIXED INTEGER BLPP TO A FORM COMPATIBLE WITH THIS PROGRAM.

*****DICTIONARY OF VARIABLES*****

XMP VARIABLES--FOR DEFINITIONS SEE THE
XMP DICTIONARY

VARIABLES USED IN BLPP BRANCH AND BOUND

BBTBIG CONTAINS VALUE OF BIGGEST PRODUCT IN TEST
FOR SELECTION OF THE BRANCHING VARIABLE.
BBITER TOTAL NUMBER OF VERTICES CREATED IN
SOLVING THE RELAXED BLPP AT EACH VERTEX OF
THE MIXED INTEGER BRANCH AND BOUND TREE.
BBTEMP CONTAINS VALUE OF PRODUCT IN TEST
FOR SELECTION OF THE BRANCHING VARIABLE.
BBVAR AN ARRAY WHICH CONTAINS THE VALUES OF

C THE VARIABLES AT THE CURRENT VERTEX OF
 C THE BRANCH AND BOUND TREE OF THE RELAXED
 C BLPP.
 C BESTAT IS THE STATUS OF EACH VARIABLE IN THE
 C INCUMBENT SOLUTION OF THE MIXED INTEGER
 C BLPP. REFER TO THE XMP DICTIONARY FOR A
 C DEFINITION OF STATUS.
 C BESTB IS THE LIST OF BASIC VARIABLES IN THE
 C INCUMBENT SOLUTION OF THE MIXED INTEGER
 C BLPP.
 C BESTLB ARRAY WHICH CONTAINS THE LOWER BOUNDS ON
 C THE VARIABLES IN THE SOLUTION OF THE
 C RELAXED BLPP.
 C BESTLO ARRAY WHICH CONTAINS THE LOWER BOUNDS ON
 C THE VARIABLES IN THE INCUMBENT SOLUTION
 C OF THE MIXED INTEGER BLPP.
 C BESTUB ARRAY WHICH CONTAINS THE UPPER BOUNDS ON
 C THE VARIABLES IN THE SOLUTION OF THE
 C RELAXED BLPP.
 C BESTUP ARRAY WHICH CONTAINS THE UPPER BOUNDS ON
 C THE VARIABLES IN THE INCUMBENT SOLUTION
 C OF THE MIXED INTEGER BLPP.
 C BIBEST ARRAY USED IN FOLLOWER'S MIXED INTEGER
 C PROBLEM. THE ARRAY CONTAINS THE VALUES
 C OF THOSE VARIABLES WHICH ARE BASIC IN THE
 C INCUMBENT SOLUTION OF THE MIXED INTEGER
 C PROBLEM.
 C BIBLO ARRAY USED IN FOLLOWER'S MIXED INTEGER
 C PROBLEM. ARRAY CONTAINS THE LOWER BOUNDS
 C ON THE VARIABLES IN THE INCUMBENT SOLUTION
 C OF THE MIXED INTEGER PROBLEM.
 C BIBSTB ARRAY USED IN FOLLOWER'S MIXED INTEGER
 C PROBLEM. ARRAY CONTAINS A LIST OF BASIC
 C VARIABLES IN THE INCUMBENT SOLUTION OF THE
 C MIXED INTEGER PROBLEM.
 C BIBUP ARRAY USED IN FOLLOWER'S MIXED INTEGER
 C PROBLEM. ARRAY CONTAINS THE UPPER BOUNDS
 C ON THE VARIABLES IN THE INCUMBENT SOLUTION
 C OF THE MIXED INTEGER PROBLEM.
 C BIFEAS FLAG USED TO INDICATE WHEN THE SOLUTION
 C OF THE RELAXED BLPP IS A SOLUTION OF THE
 C MIXED INTEGER BLPP. 0 MEANS TRUE AND
 C 1 MEANS FALSE.
 C BILEV CONTAINS THE LEVEL OF THE CURRENT VERTEX
 C IN FOLLOWER'S MIXED INTEGER BRANCH AND

C BOUND TREE.
 C BILO ARRAY USED IN FOLLOWER'S MIXED INTEGER
 C PROBLEM. STORES THE LOWER BOUND OF THE
 C SEPARATION VARIABLE AT EACH LEVEL OF THE
 C BRANCH AND BOUND TREE.
 C BIPLOR ARRAY USED IN FOLLOWER'S MIXED INTEGER
 C PROBLEM. THE ARRAY INDICATES IF THE
 C ALTERNATIVE VERTEX AT EACH LEVEL OF THE
 C MIXED INTEGER BRANCH AND BOUND TREE HAS
 C BEEN EXPLORED. 0 MEANS ALTERNATIVE NOT
 C EXPLORED AND 1 MEANS ALTERNATIVE EXPLORED.
 C BISEP ARRAY USED IN FOLLOWER'S MIXED INTEGER
 C PROBLEM. THE ARRAY TRACKS THE SEPARATION
 C VARIABLE AT EACH LEVEL OF THE BRANCH AND
 C BOUND TREE OF THE MIXED INTEGER PROBLEM.
 C BISTAT ARRAY USED IN THE FOLLOWER'S MIXED INTEGER
 C PROBLEM. THE ARRAY CONTAINS THE STATUS OF
 C EACH VARIABLE IN THE INCUMBENT SOLUTION OF
 C THE MIXED INTEGER PROBLEM. REFER TO THE
 C XMP DICTIONARY FOR A DEFINITION OF STATUS.
 C BIUP ARRAY USED IN THE FOLLOWER'S MIXED INTEGER
 C PROBLEM. STORES THE UPPER BOUND OF THE
 C SEPARATION VARIABLE AT EACH LEVEL OF THE
 C BRANCH AND BOUND TREE.
 C BIZBST VALUE OF INCUMBENT SOLUTION OF THE
 C FOLLOWER'S MIXED INTEGER PROBLEM.
 C BLVAR POINTS TO POSITION WHERE KUHN-TUCKER
 C MULTIPLIERS OF UPPER BOUND ON VARTYP = 3
 C VARIABLES IS FOUND IN BBVAR.
 C BSTNOD VERTEX AT WHICH INCUMBENT SOLUTION OF THE
 C MIXED INTEGER BLPP IS FOUND.
 C CJ1 ARRAY WHICH STORES THE COEFFICIENTS OF THE
 C LEADER'S OBJECTIVE FUNCTION.
 C CONX NUMBER OF CONSTRAINTS WHICH DO NOT CONTAIN
 C ANY VARIABLES CONTROLLED BY THE FOLLOWER.
 C CONY NUMBER OF CONSTRAINTS WHICH CONTAIN
 C VARIABLES CONTROLLED BY THE FOLLOWER.
 C CRITV IS THE NUMBER OF COMPLEMENTARY SLACKNESS
 C CONDITIONS WHICH MUST BE SATISFIED BY
 C BY THE SOLUTION OF THE BLPP.
 C DONEIT USED AS A FLAG TO INDICATE COMPLETION OF
 C MIXED INTEGER BRANCH AND BOUND PROCESS.
 C 0 MEANS NOT DONE AND 1 MEANS DONE.
 C DUMCNT USED TO KEEP TRACK OF WHEN AN UPPER
 C BOUNDED VARIABLE IS BRANCHED ON IN THE

C RELAXED BLPP.
 C FIXBAS IS THE LIST OF BASIC VARIABLES IN THE
 C SOLUTION OF THE RELAXED BLPP.
 C FIXSTA IS THE STATUS OF EACH VARIABLE IN THE
 C SOLUTION OF THE RELAXED BLPP. REFER TO
 C THE XMP DICTIONARY FOR A DEFINITION OF
 C THE STATUS ARRAY.
 C FIXZER THE ARRAY CONTAINING THE VALUES OF THOSE
 C VARIABLES WHICH ARE BASIC IN THE INCUMBENT
 C SOLUTION OF THE RELAXED BLPP.
 C INTLEV CONTAINS THE LEVEL OF THE CURRENT VERTEX
 C OF THE MIXED INTEGER BLPP BRANCH AND
 C BOUND TREE.
 C INTNOD NUMBER OF VERTEXES IN MIXED INTEGER BLPP
 C BRANCH AND BOUND TREE.
 C IVARU USED TO KEEP TRACK OF WHICH UPPER BOUNDED
 C VARIABLE IS BRANCHED ON AT EACH .
 C LEVEL CONTAINS THE LEVEL OF THE CURRENT VERTEX
 C OF THE RELAXED BLPP BRANCH AND BOUND TREE.
 C LOWER AN ARRAY WHICH KEEPS TRACK OF THE LOWER
 C BOUND ON AN INTEGER VARIABLE AT EACH LEVEL
 C OF THE MIXED INTEGER BRANCH AND BOUND
 C TREE.
 C LOWLY ARRAY WHICH STORES LOWER BOUNDS ON THE
 C VARIABLES CONTROLLED BY THE FOLLOWER.
 C USED IN SECTION WHICH DETERMINES IF
 C CURRENT VERTEX IN BRANCH AND BOUND TREE
 C CAN BE FATHOMED THROUGH BOUNDING.
 C MAXNDS MAXIMUM NUMBER OF VERTICES ALLOWED IN THE
 C BRANCH AND BOUND TREE OF THE MIXED INTEGER
 C BLPP.
 C NODE COUNTS NUMBER OF VERTICES IN THE BRANCH
 C AND BOUND TREE.
 C NODBST STORE VERTEX AT WHICH INCUMBENT SOLUTION
 C FOUND.
 C NSTOPV COUNTER USED IN LOOP USED TO SELECT THE
 C VARIABLE ON WHICH TO BRANCH.
 C NUMX NUMBER OF VARIABLES CONTROLLED BY THE
 C LEADER.
 C NUMXY NUMBER OF VARIABLES IN THE MIXED INTEGER
 C BLPP.
 C NUMY NUMBER OF VARIABLES CONTROLLED BY THE
 C FOLLOWER.
 C OKBAS ARRAY WHICH STORES THE MOST CURRENT
 C FEASIBLE BASIS.

AD-A196 113

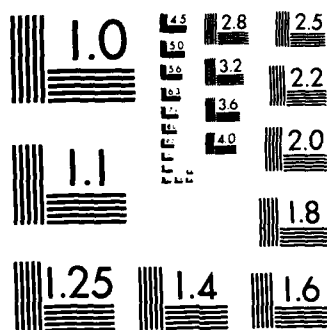
EXTENSIONS TO THE MULTILEVEL PROGRAMMING PROBLEM(U) AIR 4/4
FORCE INST OF TECH WRIGHT-PATTERSON AFB OH J T MOORE
MAY 88 AFIT/CI/NR-88-123

UNCLASSIFIED

F/G 12/5

NL

1.00
1.00
1.00
1.00
1.00



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

C OKSTAT ARRAY WHICH STORES STATUS OF EACH VARIABLE
 C IN THE MOST CURRENT FEASIBLE BASIS.
 C PAIR AN ARRAY USED TO TRACK THE THE KUHN-TUCKER
 C MULTIPLIERS ASSOCIATED WITH UPPER BOUNDS
 C ON AN INTEGER VARIABLE.
 C RANK AN ARRAY USED TO STORE THE INTEGER
 C VARIABLES IN A SPECIFIED ORDER.
 C SEPVAR IS AN ARRAY WHICH INDICATES WHICH
 C VARIABLE ARE BRANCHED ON IN THE MIXED
 C INTEGER BLPP. A NEGATIVE VALUE MEANS THE
 C VARIABLE HAS HAD LOWER BOUND PLACED ON
 C IT, AND A POSITIVE VALUE MEANS A VARIABLE
 C HAS HAD A UPPER BOUND PLACED ON IT.
 C SETU ARRAY USED TO INDICATE WHICH KUHN-TUCKER
 C MULTIPLIERS ARE ASSOCIATED WITH INTEGER
 C VARIABLES CONTROLLED BY THE FOLLOWER AND
 C MUST BE USED IF THEY BECOME UPPER BOUNDED
 C IN THE TREE.
 C UJO ARRAY WHICH STORES THE UPPER BOUNDS OF
 C VARIABLES WHICH HAVE UPPER BOUNDS.
 C UPLY ARRAY WHICH STORES UPPER BOUNDS ON THE
 C VARIABLES CONTROLLED BY THE FOLLOWER.
 C USED IN SECTION WHICH DETERMINES IF
 C CURRENT VERTEX IN BRANCH AND BOUND TREE
 C CAN BE FATHOMED THROUGH BOUNDING.
 C UPPER AN ARRAY WHICH KEEPS TRACK OF THE UPPER
 C BOUND ON AN INTEGER VARIABLE AT EACH LEVEL
 C OF THE MIXED INTEGER BRANCH AND BOUND
 C TREE.
 C VARTYP INDICATES THE VARIABLE TYPE.
 C 0 MEANS A VARIABLE IS REAL VALUED;
 C 1 MEANS A VARIABLE IS INTEGER VALUED;
 C 2 MEANS A VARIABLE HAS AN UPPER BOUND AND
 C IS INTEGER VALUED;
 C 3 MEANS A VARIABLE HAS AN UPPER BOUND AND
 C IS REAL VALUED.
 C VCNT A COUNT ON NUMBER OF VARIABLES WHICH ARE
 C NOT UNBOUNDED REAL VALUED VARIABLES
 C (VARTYP = 0).
 C VXPLOR AN ARRAY WHICH INDICATES IF THE
 C ALTERNATIVE VERTEX AT EACH LEVEL OF THE
 C MIXED INTEGER BLPP BRANCH AND BOUND TREE
 C HAS BEEN EXPLORED. 0 MEANS ALTERNATIVE
 C NOT EXPLORED AND 1 MEANS ALTERNATIVE
 C EXPLORED.

```

C      W      IS AN ARRAY WHICH INDICATES WHICH
C      COMPLEMENTARY SLACKNESS CONDITIONS HAVE
C      BEEN FORCED TO BE SATISFIED IN THE BRANCH
C      AND BOUND TREE OF THE RELAXED BLPP. A
C      NEGATIVE VALUE MEANS THE KUHN-TUCKER
C      MULTIPLIER IS SET TO ZERO, AND A
C      POSITIVE VALUE MEANS A CONSTRAINT IS
C      FORCED TO BE TIGHT.
C      WORK    A WORKING ARRAY.
C      XBEST   THE ARRAY CONTAINING THE VALUES OF THOSE
C      VARIABLES WHICH ARE BASIC IN THE INCUMBENT
C      SOLUTION OF THE MIXED INTEGER BLPP.
C      XINT    NUMBER OF INTEGER VARIABLE CONTROLLED BY
C      THE LEADER.
C      YCJ     THE ARRAY CONTAINING THE COEFFICIENTS OF
C      THE VARIABLES IN THE FOLLOWER'S OBJECTIVE
C      FUNCTION.
C      YINT    NUMBER OF INTEGER VARIABLE CONTROLLED BY
C      THE FOLLOWER.
C      ZBEST   VALUE OF THE INCUMBENT SOLUTION OF THE
C      MIXED INTEGER BLPP.
C      ZMOST   VALUE OF THE SOLUTION OF THE RELAXED BLPP.
C      *****END OF DEFINITIONS*****
C
C      *****NOTE*****
C      X REFERS TO VARIABLES CONTROLLED BY THE LEADER;
C      Y REFERS TO VARIABLES CONTROLLED BY THE FOLLOWER;
C      U REFERS TO KUHN-TUCKER MULTIPLIERS ASSOCIATED
C      WITH CONSTRAINTS CONTAINING VARIABLES
C      CONTROLLED BY THE FOLLOWER AND WITH THE
C      NONNEGATIVE RESTRICTIONS ON THE VARIABLES
C      CONTROLLED BY THE FOLLOWER;
C      G REFERS TO THE SLACK OR SURPLUS VARIABLE
C      ASSOCIATED WITH EACH CONSTRAINT CONTAINING
C      VARIABLES CONTROLLED BY THE FOLLOWER.
C      *****END OF NOTE*****
C
C      SUMMARY OF INPUT FORMATS:
C
C      CRITV, NUMX, NUMY, CONX, CONY, XINT, YINT, MAXNDS      815
C
C      M, NSTRUC, MINMAX, ECHO, PRINT, BNDTYP                615
C
C      (I, ROWTYP(I), B(I))  I=1,M                          215,F13.3
C

```

```

C      XSUB,J,COLLEN,CJ,UJ,LJ,
C      VARTYP(J)          A4,2I5,F15.5,2F10.0,I2
C
C      (I, A(I,J)) FOR I WITH A(I,J) NONZERO  I5,F15.5
C
C      INTEGER TOTCP1,VIRCP1,TOTCP2,VIRCP2,TOTCP,
1      VIRCP,VARCP
C
C      *****DECLARE XMP VARIABLES
C      DOUBLE PRECISION B(100),BASCB(100),BASLB(100),
1      BASUB(100),BOUND,CJ,COLA(50),
2      LJ,MEMR(6000),UJ,UZERO(100),
3      XBZERO(100),YQ(100),Z
C
C      IF YOU HAVE ANY TWO-SIDED CONSTRAINTS (ROWTYP=2),
C      THEN DIMENSION BLOW(MAXM), THE SAME AS B(MAXM).
C      OTHERWISE, JUST DIMENSION BLOW(1).
C      NOTE: THIS VERSION OF XDEMO DOES NOT ALLOW
C      TWO-SIDED CONSTRAINTS.
C      DOUBLE PRECISION BLOW(1)
C
C      INTEGER ARRAYS AND VARIABLES.
C      INTEGER BNDTYP,COLLEN,COLMAX,ERROR,
1      FACTOR,IOERR,IOLOG,ITER,ITER1,
2      ITER2,LOOK,M,MAPI(9),MAPR(8),
3      MAXA,MAXM,MAXN,N,NTYPE2,PICK,
4      PRINT,TERMIN,UNBDDQ
C      INTEGER MEMI(6000)
C      INTEGER BASIS(100),COLI(50),ROWTYP(100),
1      STATUS(600)
C
C      DOUBLE PRECISION CJTEMP,ZTEMP,UTEMP
C      INTEGER ECHO,IOIN,ISEQ,JSEQ,JSEQ2,NSTRUC,MINMAX
C      CHARACTER*4 DUMY
C      CHARACTER*4 XSUB
C
C      *****DECLARE VARIABLES NEEDED FOR BLPP BRANCH
C      AND BOUND
C      DOUBLE PRECISION BBVAR(100),XBEST(100),ZBEST,
1      BTEMP,BBTBIG,WORK(600),
2      BIGINT,UPLY(100),LOWLY(100),DIFF

```

```

      INTEGER LEVEL, CRITV, W(100), BESTB(100), BESTAT(600),
1      RANK(100), L, IX, BBITER, NUMX, NUMY,
2      CONX, CONY, NSTOPV, NUMX1, NUMXY, FLGUP,
3      FIXBAS(100), FIXSTA(600), BLVAR, CCNXY,
4      BSTCP, BSTNOD
      DOUBLE PRECISION UPPER(999), LOWER(999), ZMOST,
1      BBTEM1, BBTEM2, BESTLO(600),
2      BESTUP(600), FIXZER(100),
3      BESTLB(600), BESTUB(600), CJ1(600),
4      UJO(600), SETU(100), YCJ(600),
5      ZBND(0:999)
      INTEGER DONEIT, INTLEV, DUMCNT, IVARU(100),
1      VARTYP(600), SEPVAR(999), VCNT, PAIR(100),
2      FLAGUP(100), OKBAS(100), OKSTAT(600), XINT,
3      YINT
C      *****XMP COMMON VARIABLES
      COMMON/XMPCOM/BIG, SMALL, ZL, ZLC,
1      EPS1, EPS2, EPS3, EPS4, EPS5, EPS6
      DOUBLE PRECISION BIG, SMALL, ZL, ZLC,
1      EPS1, EPS2, EPS3, EPS4, EPS5, EPS6
      COMMON/XMPLEN/LENI, LENMI, LENMR, LENR
      INTEGER LENI, LENMI, LENMR, LENR

C
C
C      DATA XSUB/'XSUB'/
C      *****SUBROUTINES CALLED
C      XMAPS, XADDAJ, XADDUB, XSLACK, XSTART
C      XPRIML, XPRINT, FIXINT
C
C      *****BODY OF PROGRAM (BLPPM1)
C
C      IOIN IS LOCAL TO THIS USER PROGRAM.  IT SPECIFIES
C      THE I/O UNIT FROM WHICH THE DATA IS TO BE READ.
C      IOERR AND IOLOG ARE XMP VARIABLES.  IOERR IS THE
C      I/O UNIT WHERE ERROR MESSAGES ARE TO BE WRITTEN.
C      IOLOG IS THE I/O UNIT WHERE LOG INFORMATION IS TO
C      BE WRITTEN, IF REQUESTED.
C
C      IOIN=5
C      IOERR=6
C      IOLOG=6
C
C      ***PRINT HEADING***
C

```

```

      WRITE(IOLOG,917)
917  FORMAT(1H1///10X,'LINEAR MIXED INTEGER BILEVEL ',
1      'PROGRAMMING PROBLEM')
C
C      ***SET XMP VARIABLES***
      MAXA=1500
      MAXM=100
      MAXN=600
      COLMAX=50
C
      PICK=6
      LOOK=200
      FACTOR=50
C
      LENI=6000
      LENR=6000
      LENMI=9
      LENMR=8
C
C      ***READ INFORMATION ON MIXED INTEGER BLPP***
      READ(IOIN,899) CRITV,NUMX,NUMY,CONX,CONY,XINT,
1      YINT,MAXNDS
899  FORMAT(8I5)
C
C      READ(IOIN,900) M,NSTRUC,MINMAX,ECHO,PRINT,BNDTYP
900  FORMAT(6I5)
C      ***INITIALIZE BRANCH AND BOUND VARIABLES***
      NUMINT=XINT+YINT
      NUMXY=NUMX+NUMY
      NUMXY1=NUMXY+1
      NUMX1=NUMX+1
      CONXY=CONX+CONY
      NSTOPV=CRITV+NUMX
      BLVAR=NUMX+NUMY+CONY+NUMY+CONY
      ZBEST=-1.0D24
      ZMOST=-1.0D24
      DONEIT=0
      INTLEV=0
      BBITER=0
      INTNOD=0
      LEVEL=0
      DO 1199 J=1,NSTRUC
      SETU(J)=0.0
1199  BBVAR(J)=0.0

```

```

      DO 1198 J=1,NSTOPV
1198 W(J)=0
C
      IF(MINMAX .EQ. 1)WRITE(IOLOG,960)
      IF(MINMAX .EQ. -1)WRITE(IOLOG,961)
960 FORMAT(1H0,10X,26HTHIS IS A MAXIMIZATION RUN//)
961 FORMAT(1H0,10X,26HTHIS IS A MINIMIZATION RUN//)
C
      WRITE(IOLOG,918)M
918 FORMAT(1H0,5X,I6,2X,11HCONSTRAINTS)
      WRITE(IOLOG,919)NSTRUC
919 FORMAT(1H0,5X,I6,2X,20HSTRUCTURAL VARIABLES//)
      WRITE (6,920) NUMX,NUMY,CONX,CONY
920 FORMAT(' NUMBER OF VARIABLES CONTROLLED BY THE ',
1         'LEADER ',I3,/' NUMBER OF VARIABLES ',
2         'CONTROLLED BY THE FOLLOWER ',I3,/'
3         ' NUMBER OF CONSTRAINTS WITH NO',
4         ' FOLLOWER VARIABLES ',I3,/'
5         ' NUMBER OF CONSTRAINTS WITH FOLLOWER ',
6         'VARIABLES ',I3)

      CALL XMAPS(BNDTYP,IOERR,MAPI,MAPR,MAXA,
1              MAXM,MAXN,MEMI,MEMR)
C
C      READ IN THE ROW TYPES AND THE RIGHT-HAND-SIDE.
C      ISEQ IS THE ROW NUMBER, FOR THE USER'S
C      CONVENIENCE.
      NOSX=NUMX
      DO 100 I=1,M
          READ(IOIN,902) ISEQ,ROWTYP(I),B(I)
          IF (I.LE.CONXY)GO TO 100
          NOSX=NOSX+1
          YCJ(NOSX)=-B(I)
100 CONTINUE
902 FORMAT(2I5,F13.3)
C
      IF(ECHO .LT. 1)GO TO 110
      WRITE(IOLOG,913)
913 FORMAT(///10X,7HROW NO.,10X,8HROW TYPE,10X,
1         16H RIGHT-HAND-SIDE/)
      DO 105 I=1,M
          WRITE(IOLOG,914) I,ROWTYP(I),B(I)
105 CONTINUE
914 FORMAT(10X,I7,10X,I8,10X,D16.8)
110 CONTINUE

```



```

C
C      N IS THE CURRENT NUMBER OF VARIABLES.
C      N IS INCREMENTED BY XADDAJ AS EACH COLUMN
C      IS ADDED.
C      N=0
C      JSEQ2=0
C
C      ***READ IN THE COLUMNS ONE AT A TIME***
C
C      JSEQ IS A SEQUENCE NUMBER FOR THE USER'S
C      CONVENIENCE
C      (XMP NUMBERS THE COLUMNS SEQUENTIALLY AS THEY ARE
C      SENT TO XADDAJ AND RETURNS THIS NUMBER IN IDNO)
C
C      INTCNT=0
C      DO 120 J=1,NSTRUC
C          READ(IOIN,904) DUMY,JSEQ,COLLEN,CJ,UJ,LJ,
C              VARTYP(J)
C          1
C          904      FORMAT(A4,2I5,F15.5,2F10.0,I2)
C              UJO(J)=UJ
C              CJ1(J)=CJ
C              WORK(J)=CJ
C              IF(DUMY .EQ. XSUB)GO TO 1105
C              WRITE(IOERR,901)JSEQ2
C          901      FORMAT(31H ERROR IN INPUT FILE AFTER XSUB,I5)
C              STOP
C          1105     JSEQ2=JSEQ
C              IF (JSEQ.LT.NUMX1 .OR. VARTYP(J).EQ.0
C              1      .OR. VARTYP(J).EQ.3) GO TO 789
C              INTCNT=INTCNT+1
C              PAIR(J)=BLVAR+INTCNT
C              IF (VARTYP(J).EQ.1) SETU(BLVAR+INTCNT)=1.0
C          789     IF (JSEQ.LE.NUMXY) BLPPCT=BLPPCT+COLLEN
C              FLIP THE SIGN OF THE OBJECTIVE FUNCTION
C              COEFFICIENT IF WE ARE MINIMIZING.
C              CJTEMP=CJ
C              IF(MINMAX .EQ. -1) CJTEMP=-CJ
C              DO 130 K=1,COLLEN
C                  READ(IOIN,905) COLI(K),COLA(K)
C          130     CONTINUE
C          905     FORMAT(I5,F15.5)
C
C              CALL XADDAJ(CJTEMP,COLA,COLI,COLLEN,COLMAX,
C              1              IOERR,IDNO,MAPI,MAPR,MEMI,MEMR,N)
C

```

```

        IF(BNDTYP .GT. 1)GO TO 112
        LJ=0.0
        UJ=BIG
        GO TO 115
112      IF(BNDTYP .GT. 2)GO TO 113
        ERROR=3
        CALL XSTOP(ERROR,IOERR)
113      IF(BNDTYP .EQ. 3) LJ=0.0
C
        IF (IDNO.GT.BLVAR .AND. SETU(IDNO).EQ.1.0)
1      UJ=0.0
        CALL XADDUB (BNDTYP,IOERR,IDNO,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
C
115      IF(ECHO .LT. 1)GO TO 120
        WRITE(IOLOG,915)IDNO,CJ,UJ,LJ
915      FORMAT(/1X,2HJ=,I5,5X,3HCJ=,D16.8,5X,
1      3HUJ=,D16.8,5X,3HLJ=,D16.8)
        DO 118 K=1,COLLEN
        WRITE(IOLOG,916) COLI(K),COLA(K)
118      CONTINUE
916      FORMAT(13X,I6,5X,D16.8)
120 CONTINUE
C
C      ***RANK ORDERS THE INTEGER VARIABLES ACCORDING
C      TO THEIR COEFFICIENT IN THE LEADER'S
C      OBJECTIVE FUNCTION***
        K=1
150 BIGINT=-1.0D20
        DO 155 I=1,NUMXY
        IF (VARTYP(I).EQ.0 .OR. VARTYP(I).EQ.3)
1      GO TO 155
        IF (WORK(I).LE.BIGINT) GO TO 155
        RANK(K)=I
        BIGINT=WORK(I)
155 CONTINUE
        WORK(RANK(K))=-1.0D25
        K=K+1
        IF (K.GT.NUMINT) GO TO 160
        GO TO 150
160 CONTINUE
        DO 161 I=1,NUMINT
        WRITE (6,162) RANK(I),I
161 CONTINUE
162 FORMAT(' VAR ',I3,' IS INTEGER WITH RANK ',I3)

```

```

C
C      ***START ALL OF THE STRUCTURAL VARIABLES AT THEIR
C      LOWER BOUNDS***
      DO 170 J=1,N
          STATUS(J)=0
170  CONTINUE
C
      CALL XSLACK(B,BASCB,BASIS,BASLB,BASUB,BLOW,
1          BNDTYP,BOUND,
2          COLA,COLI,COLMAX,IOERR,
3          M,MAPI,MAPR,MAXM,MAXN,MEMI,
4          MEMR,N,ROWTYP,STATUS,
5          UZERO,XBZERO,Z)
C
      CALL CPTIME(TOTCP1,VIRCP1)
      CNT=1.0
      GO TO 4500
C
C      ***BEGIN BRANCH AND BOUND***
1998  CALL FIXINT(ZMOST,ZBEST,NUMX,NUMY,VARTYP,UJO,
1          YCJ,LOWER,UPPER,XBEST,BESTB,BESTAT,
2          BESTLO,BESTUP,SEPVAR,DONEIT,PAIR,
3          BLVAR,NUMXY,NUMX1,CJ1,INTNOD,RANK,
4          NUMINT,BESTUB,BESTLB,CONX,CONY,
5          NSTRUC,INTLEV,BSTCP,BSTNOD,B,BASCB,
6          FIXBAS,BASLB,BASUB,BLOW,ROWTYP,
7          BNDTYP,BOUND,COLA,COLI,COLMAX,
8          IOERR,M,MAPI,MAPR,MAXM,MAXN,
9          MEMI,MEMR,N,NTYPE2,SCODE,FIXSTA,
0          UZERO,FIXZER,Z,FACTOR,IOLOG,
1          ITER1,ITER2,LOOK,PICK,PRINT,
2          TERMIN,UNBDDQ,YQ)
2301  CONTINUE
      IF (DONEIT.EQ.1) GO TO 6000
      IF (INTNOD.LT.MAXNDS) GO TO 2304
      FLGOUT=1.0
      GO TO 6000
C
C      ***CHECK TO SEE IF THE CURRENT NODE CAN BE
C      FATHOMED BY BOUNDING***
2304  ZMOST=-1.0D25
      CNT=0.0
      DO 2310 I=NUMX1,NUMXY
          CALL XGETUB(BNDTYP,IOERR,I,LJ,MAPI,MAPR,
1              MEMI,MEMR,UJ)
          LOWLY(I)=LJ

```

```

        UPLY(I)=UJ
        LJ=0.0
        UJ=UJO(I)
        CALL XADDUB(BNDTYP,IOERR,I,LJ,MAPI,MAPR,
1          MEMI,MEMR,UJ)
2310 CONTINUE
        GO TO 4000
2320 DO 2330 I=NUMX1,NUMXY
        LJ=LOWLY(I)
        UJ=UPLY(I)
        CALL XADDUB(BNDTYP,IOERR,I,LJ,MAPI,MAPR,
1          MEMI,MEMR,UJ)
2330 CONTINUE
        IF (TERMIN.NE.1) ZTEMP=-1.0D26
        ZBND(INTNOD)=ZTEMP
        DIFF=ZBND(INTNOD)-ZBEST
        IF (DIFF.LE. 0.000001) GO TO 1998
        GO TO 4000
1999 CONTINUE
C      ***DETERMINE VALUE OF EACH VARIABLE IN
C      CURRENT SOLUTION***
        DO 670 I=NUMX1,NSTRUC
        IF (STATUS(I)) 400,500,600
400      IF (STATUS(I).EQ.-2 .OR. STATUS(I).EQ.-3)
1        GO TO 430
        IF (STATUS(I).EQ.-4) GO TO 500
        CALL XGETUB(BNDTYP,IOERR,I,LJ,MAPI,MAPR,
          MEMI,MEMR,UJ)
        BBVAR(I)=UJ
        GO TO 670
430      DO 440 J=1,M
          IF(BASIS(J).NE.I) GO TO 440
          BBVAR(I)=XBZERO(J)
          GO TO 670
440      CONTINUE
500      IF (BNDTYP.NE.4) GO TO 555
        CALL XGETUB(BNDTYP,IOERR,I,LJ,MAPI,MAPR,
1          MEMI,MEMR,UJ)
        IF (LJ.EQ.0.0) GO TO 555
        BBVAR(I)=LJ
        GO TO 670
555      BBVAR(I)=0.0
        GO TO 670
600      IX=STATUS(I)
        BBVAR(I)=XBZERO(IX)

```

```

670 CONTINUE
C   ***FEASIBILITY CHECK/SELECT BRANCHING VARIABLE***
C   THIS SECTION CHECKS TO SEE IF U*G=0 FOR EACH
C   CONSTRAINT ON Y.  IF YES, THEN FEASIBLE SOLUTION
C   FOUND.  IF NO, THEN PICK LARGEST VALUE
C   OF U*G AND GO TO BRANCH ON U(IE MAKE U ZERO)
C   FLAG=1 MEANS NORMAL WITH NO WORRY ABOUT INTEGERS;
C   FLAG=2 MEANS WORKING WITH Y VAR AND LOWER BOUND U;
C   FLAG=3 MEANS WORKING WITH Y VAR AND ITS UPPER
C   BOUND U.
C
  BBTBIG=0.00001
  VCNT=0
  FLGUP=1
  DO 1200 I=NUMX1,NSTOPV
    IF (VARTYP(I).EQ.0) GO TO 1210
    CALL XGETUB(BNDTYP,IOERR,I,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
    VCNT=VCNT+1
    IF (UJ.GE.UJO(I)) GO TO 1205
    IF (VARTYP(I).LT.3 .AND. UJ.EQ.BBVAR(I))
1      GO TO 1200
    BBTEM2=(UJ-BBVAR(I))*BBVAR(BLVAR+VCNT)
    IF (BBTEM2.LE.BBTBIG) GO TO 1205
    BBTBIG=BBTEM2
    KK=I
    DUMCNT=VCNT
    FLGUP=3
1205  BBTEM1=(BBVAR(I)-LJ)*BBVAR(I+CRITV)
    IF (BBTEM1.LE.BBTBIG) GO TO 1200
    BBTBIG=BBTEM1
    KK=I
    FLGUP=2
    GO TO 1200
1210  BBTEMP=BBVAR(I)*BBVAR(I+CRITV)
    IF (BBTEMP.LE.BBTBIG) GO TO 1200
    BBTBIG=BBTEMP
    KK=I
    FLGUP=1
1200  CONTINUE
    IF (BBTBIG.LE.0.00001) GO TO 2000
C   ***BRANCH ON SELECTED VARIABLE***
C   BRANCHES ON SELECTED U VARIABLE(IE SETS U TO 0)
C   W-ARRAY USED TO TELL WHICH VARIABLE OF A U-G PAIR
C   IS SET.  W(I)=K MEANS VARIABLE K SET AT LEVEL I OF

```

```

C      TREE.  IF K<0, THAT MEANS ALTERNATIVE OF U-G
C      PAIR NOT CONSIDERED YET.
1500  LEVEL=LEVEL+1
      FLAGUP(LEVEL)=FLGUP
      GO TO (1510,1510,1520) FLAGUP(LEVEL)
1510  W(LEVEL)=-KK
      L=KK+CRITV
      UJ=0.0
      LJ=0.0
      CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
      GO TO 4000
1520  W(LEVEL)=-KK
      L=BLVAR+DUMCNT
      UJ=0.0
      LJ=0.0
      IVARU(LEVEL)=DUMCNT
      CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
      GO TO 4000
C      ***UPDATE INCUMBENT SOLUTION OF RELAXED BLPP***
2000  ZMOST=ZTEMP
      IF(ZMOST.GT.1.0D5) GO TO 5000
      DO 2100 K=1,M
          FIXZER(K)=XBZERO(K)
          FIXBAS(K)=BASIS(K)
2100  CONTINUE
      DO 2200 K=1,N
          FIXSTA(K)=STATUS(K)
2200  CONTINUE
      DO 2250 K=1,NSTRUC
          CALL XGETUB(BNDTYP,IOERR,K,LJ,MAPI,MAPR,
1          MEMI,MEMR,UJ)
          BESTLB(K)=LJ
          BESTUB(K)=UJ
2250  CONTINUE
      IF (LEVEL.EQ.0) GO TO 1998
      GO TO 3100
2500  IF (TERMIN.EQ.2) GO TO 5000
      IF (TERMIN.NE.1 .AND. BBITER.EQ.0) GO TO 5000
C      ***BACKTRACK***
C      FINDS A LIVE NODE, FREES U VAR, SETS G VAR TO 0
C      ALSO FREES VARIABLES WHICH HAVE HAD BOTH
C      ALTERNATIVES CONSIDERED.
3100  CONTINUE

```

```

IF (LEVEL.EQ.0) GO TO 1998
IF(W(LEVEL).LT.0) GO TO 3200
L=W(LEVEL)
UJ=UJO(L)
LJ=0.0
IF (VARTYP(L).EQ.0 .OR. VARTYP(L).EQ.3) GO TO 3120
IF (INTLEV.EQ.0) GO TO 3120
DO 3110 K=1,INTLEV
    IF (ABS(SEPVAR(K)).NE.L) GO TO 3110
    LJ=LOWER(K)
    UJ=UPPER(K)
3110 CONTINUE
3120 CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
    W(LEVEL)=0
    LEVEL=LEVEL-1
    GO TO 3100
3200 GO TO (3210,3220,3230) FLAGUP(LEVEL)
C ***RESET U OF A CONTINUOUS UNBOUNDED VARIABLE***
3210 L=-W(LEVEL)+CRITV
    UJ=UJO(L)
    LJ=0.0
    CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
C ***SET A CONT Y OR G TO ZERO***
    W(LEVEL)=-W(LEVEL)
    L=W(LEVEL)
    UJ=0.0
    LJ=0.0
    CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
GO TO 4000
C ***WORK WITH INTEGER Y AND ITS LOWER BOUND U***
3220 L=-W(LEVEL)+CRITV
C RESET THE U
    UJ=UJO(L)
    LJ=0.0
    CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
C ***SET Y TO ZERO OR A PREVIOUS SET LOWER BOUND***
    W(LEVEL)=-W(LEVEL)
    L=W(LEVEL)
    LJ=0.0
    UJ=0.0
    IF (INTLEV.EQ.0 .OR. VARTYP(L).EQ.3) GO TO 3226

```

```

DO 3225 K=1,INTLEV
  IF (ABS(SEPVAR(K)).NE.L) GO TO 3225
  LJ=LOWER(K)
  UJ=LJ
3225 CONTINUE
3226 CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
  GO TO 4000
C    ***WORK WITH INTEGER Y AND ITS UPPER BOUND U
C      OR WITH CONTINUOUS BOUNDED Y AND ITS UPPER
C      BOUND***
3230 L=BLVAR+IVARU(LEVEL)
C    RESET THE U
  UJ=UJO(L)
  LJ=0.0
  CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
C    ***SET Y TO ITS UPPER BOUND OR TO A PREVIOUSLY SET
C      UPPER BOUND***
  W(LEVEL)=-W(LEVEL)
  L=W(LEVEL)
  UJ=UJO(L)
  LJ=UJO(L)
  IF (INTLEV.EQ.0 .OR. VARTYP(L).EQ.3) GO TO 3236
DO 3235 K=1,INTLEV
  IF (ABS(SEPVAR(K)).NE.L) GO TO 3235
  UJ=UPPER(K)
  LJ=UJ
3235 CONTINUE
3236 CALL XADDUB(BNDTYP,IOERR,L,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
C    ***USE THE XMP SUBROUTINES TO SOLVE A LINEAR
C      PROGRAMMING PROBLEM USING THE PRIMAL
C      SIMPLEX METHOD***
4000 DO 4001 K=1,M
  BASIS(K)=OKBAS(K)
4001 CONTINUE
  DO 4002 K=1,N
  STATUS(K)=OKSTAT(K)
4002 CONTINUE

```



```

      CALL XSTART(B,BASCB,BASIS,BASLB,BASUB,
1          BNDTYP,BOUND,
2          COLA,COLI,COLMAX,IOERR,
3          M,MAPI,MAPR,MAXM,MAXN,MEMI,
4          MEMR,N,NTYPE2,SCODE,STATUS,
5          UZERO,XBZERO,Z)
C
      BBITER=BBITER+1
C
4500 CALL XPRIML(B,BASCB,BASIS,BASLB,BASUB,BNDTYP,
1          BOUND,COLA,COLI,COLMAX,
2          FACTOR,IOERR,IOLOG,ITER1,ITER2,
3          ITER2,LOOK,M,MAPI,MAPR,MAXM,MAXN,
4          MEMI,MEMR,N,NTYPE2,PICK,PRINT,
5          STATUS,TERMIN,UNBDDQ,UZERO,
6          XBZERO,YQ,Z)
C
C      FLIP THE SIGN OF THE OBJECTIVE FUNCTION VALUE IF
C      WE ARE MINIMIZING.
      ZTEMP=Z
      IF(MINMAX .EQ. -1) ZTEMP=-ZTEMP
C
      CNT=CNT+1.0
      IF (CNT.EQ.1) GO TO 2320
      IF(TERMIN.NE.1) GO TO 2500
      DO 4010 K=1,M
          OKBAS(K)=BASIS(K)
4010 CONTINUE
      DO 4020 K=1,N
          OKSTAT(K)=STATUS(K)
4020 CONTINUE
      IF (BBITER.EQ.0) GO TO 2301
      IF (ZTEMP.LE.ZMOST) GO TO 3100
      GO TO 1999
C
C      THE MIXED INTEGER BLPP HAS NO FEASIBLE SOLUTION
C      OR HAS AN UNBOUNDED SOLUTION.
C
5000 WRITE(IOLOG,7000)
7000 FORMAT(1H1///,10X,'THE PROBLEM WENT BOOM ',
1          'WITH A CRASH.')
      GO TO 314
C
C      ***PRINT OUT THE SOLUTION***
C

```

```

6000 CALL CPTIME(TOTCP2,VIRCP2)
      IF (FLGOUT.EQ.1.0) WRITE (6,1984)
1984 FORMAT(' THE FOLLOWING SOLUTION IS NOT OPTIMAL.',
1       ' MAX NODES EXCEEDED')
      CALL XPRINT(BESTB,BNDTYP,BOUND,
1       IOERR,IOLOG,NUMXY,M,
2       MAPI,MAPR,MAXM,MAXN,MEMI,MEMR,
3       N,NTYPE2,BESTAT,XBEST,ZBEST,
4       BESTLO,BESTUP)
C
      IF (FLGOUT.EQ.1.0) WRITE (6,1985)
1985 FORMAT(' THE ABOVE SOLUTION IS NOT OPTIMAL.',
1       ' MAX NODES EXCEEDED')
      WRITE(IOLOG,1986)BBITER
1986 FORMAT(5X,'THE NUMBER OF B&B ITERATIONS IS',I7)
      WRITE (6,1987) INTNOD
1987 FORMAT(' THE NUMBER OF INTEGER NODES IS',I6)
      IBSTCP=BSTCP-TOTCP1
      WRITE (6,947) BSTNOD,IBSTCP
947 FORMAT(' BEST SOLUTION AT NODE ',I5,
1       ' AT TIME ',I8)
C
314 CONTINUE
C
      TOTCP=TOTCP2-TOTCP1
      VIRCP=VIRCP2-VIRCP1
      WRITE (6,70) TOTCP,VIRCP
70  FORMAT(3X,' TOTAL CPTIME =',I9,3X,
1       'VIRTUAL CPTIME =',I9)
      STOP
      END
      SUBROUTINE FIXINT(ZMOST,ZBEST,NUMX,NUMY,VARTYP,
1       UJO,YCJ,LOWER,UPPER,XBEST,
2       BESTB,BESTAT, BESTLO,BESTUP,
3       SEPVAR,DONEIT,PAIR,BLVAR,
4       NUMXY,NUMX1,CJ1,INTNOD,RANK,
5       NUMINT,BESTUB,BESTLB,CONX,CONY,
6       NSTRUC,INTLEV,BSTCP,BSTNOD,B,
7       BASCB,FIXBAS,BASLB,BASUB,BLOW,
8       ROWTYP,BNDTYP,BOUND,COLA,COLI,
9       COLMAX,IOERR,M,MAPI,MAPR,MAXM,
0       MAXN,MEMI,MEMR,N,NTYPE2,SCODE,
1       FIXSTA,UZERO,FIXZER,Z,FACTOR,
2       IOLOG,ITER1,ITER2,LOOK,PICK,
3       PRINT,TERMIN,UNBDDQ,YQ)

```

```

C      *****PURPOSE
C      THIS SUBROUTINE PERFORMS THE BRANCH AND BOUND
C      PROCESS ON THE INTEGER VARIABLES OF THE MIXED
C      INTEGER BLPP. ALTERNATIVE CONSTRAINTS ARE IMPOSED
C      BY PLACING UPPER AND LOWER BOUNDS ON THE
C      VARIABLES.
C
C      *****DECLARE XMP VARIABLES
C      INTEGER COLMAX,MAXM,MAXN,PICK
C      DOUBLE PRECISION XBZERO(MAXM),LJ,UJ,MEMR(LENR),
1          B(MAXM),BASCB(MAXM),BASLB(MAXM),
2          BASUB(MAXM),BOUND,COLA(COLMAX),
3          UZERO(MAXM),Z,BLOW(1),YQ(MAXM)
C      INTEGER FACTOR,IOLOG,ITER1,ITER2,LOOK,PRINT,
1          TERMIN,UNBDDQ,BASIS(MAXM),M,N,
2          STATUS(MAXN),MAPI(LENMI),MAPR(LENMR),
3          MEMI(LENI),BNDTYP,IOERR,NSTRUC,NTYPE2,
4          SCODE,COLI(COLMAX),ROWTYP(MAXM)
C      *****DECLARE BLPP VARIABLES
C      DOUBLE PRECISION LOWER(999),UPPER(999),ZMOST,
1          XBEST(100),UJO(600),YCJ(600),
2          BESTLO(600),BESTUP(600),CHEC,
3          CJ1(600),DIFFR,UP,LO,ZBEST,
4          BESTUB(600),BESTLB(600)
C      INTEGER BIFEAS,NUMXY,VARTYP(600),WORK(100),CONX,
1          CONY,INTLEV,BESTB(100),NUMX1,IB,IBAS,
2          BESTAT(MAXN),SEPVAR(999),BSTCP,BSTNOD,
3          DONEIT,IXBAS,VXPLOR(999),RANK(100),
4          PAIR(100),NUMX,BLVAR,NUMY
C      *****COMMON VARIABLES
C      COMMON/XMPCOM/BIG,SMALL,ZL,ZLC,EPS1,
1          EPS2,EPS3,EPS4,EPS5,EPS6
C      DOUBLE PRECISION BIG,SMALL,ZL,ZLC,
1          EPS1,EPS2,EPS3,EPS4,EPS5,EPS6
C      COMMON/XMPLEN/LENI,LENMI,LENMR,LENR
C      INTEGER LENI,LENMI,LENMR,LENR
C
C      IF (ZMOST.LE.-1.0D20) GO TO 2000
C*****
C*      ABOVE LINE USED IN ALGORITHMS 1, 2, 3, AND 4      *
C*      FOR ALGORITHMS 5, 6, 7, AND 8 SUBSTITUTE        *
C*      FOLLOWING LINES:                                  *
C*      DIFF=ZMOST-ZBEST                                  *
C*      IF (DIFF.LE.0.000001) GO TO 2000                  *
C*****

```

```

C      ***SELECT A BRANCHING VARIABLE***
      CHEC=-100000.
      IB=0
      DO 100 I=1,NUMXY
        IF (VARTYP(I).EQ.0 .OR. VARTYP(I).EQ.3)
1          GO TO 100
        IF (STATUS(I).LE.0) GO TO 100
        JJ=STATUS(I)
        IBAS=DINT(XBZERO(JJ))
        DIFFR=XBZERO(JJ)-DFLOAT(IBAS)
        IF (DIFFR.LE.0.000001 .OR. DIFFR.GT.0.999999)
1          GO TO 100
        IF (DIFFR.LT.CHEC) GO TO 100
        IB=I
        IXBAS=IBAS
        CHEC=DIFFR
100    CONTINUE
        IF (IB.EQ.0) GO TO 110
C      ***BRANCH ON SELECTED VARIABLE***
101    CALL XGETUB(BNDTYP,IOERR,IB,LJ,MAPI,MAPR,
1          MEMI,MEMR,UJ)
        IF (CJ1(IB).GT.0.0) GO TO 105
102    UJ=DFLOAT(IXBAS)
        INTLEV=INTLEV+1
        SEPVAR(INTLEV)=IB
        IF (IB.LE.NUMX) GO TO 107
        LO=0.0
        UP=UJO(PAIR(IB))
        CALL XADDUB(BNDTYP,IOERR,PAIR(IB),LO,MAPI,MAPR,
1          MEMI,MEMR,UP)
        GO TO 107
105    LJ=DFLOAT(IXBAS)+1.0
        INTLEV=INTLEV+1
        SEPVAR(INTLEV)=-IB
107    CALL XADDUB(BNDTYP,IOERR,IB,LJ,MAPI,MAPR,
1          MEMI,MEMR,UJ)
        INTNOD=INTNOD+1
        VXPLOR(INTLEV)=0
        LOWER(INTLEV)=LJ
        UPPER(INTLEV)=UJ
        GO TO 3999

```

```

C      ***CHECK TO SEE IF AT A BIFEASIBLE POINT***
110 CALL BILEVF (NUMX,NUMY,VARTYP,BIFEAS,UJO,YCJ,
1          BESTUB,BESTLB,CONX,CONY,NSTRUC,CJ1,
2          BSTCP,BSTNOD,XBEST,BESTB,BESTAT,
3          BESTLO,BESTUP,ZBEST,INTNOD,B,BASCB,
4          BASIS,BASLB,BASUB,BLOW,ROWTYP,
5          BNDTYP,BOUND,COLA,COLI,COLMAX,IOERR,
6          M,MAPI,MAPR,MAXM,MAXN,MEMI,MEMR,
7          N,NTYPE2,SCODE,STATUS,UZERO,XBZERO,
8          Z,FACTOR,IOLOG,ITER1,ITER2,LCOK,
9          PICK,PRINT,TERMIN,UNBDDQ,YQ)
      GO TO 3500
C*****
C*      ABOVE LINE USED IN ALGORITHMS 1 AND 5      *
C*      FOR ALGORITHMS 2 AND 6 SUBSTITUTE          *
C*      FOLLOWING LINE:                            *
C*      IF (BIFEAS.NE.2) GO TO 3500                *
C*      FOR ALGORITHMS 3 AND 7 SUBSTITUTE          *
C*      FOLLOWING LINE:                            *
C*      IF (BIFEAS.EQ.1) GO TO 3500                *
C*      FOR ALGORITHMS 4 AND 8 SUBSTITUTE          *
C*      FOLLOWING LINE:                            *
C*      CONTINUE                                    *
C*****
C      ***BACKTRACK***
2000 IF (INTLEV.EQ.0) GO TO 4000
      IF (VXPLOE(INTLEV).EQ.0) GO TO 3000
      LL=ABS(SEPVAR(INTLEV))
      LJ=0.0
      UJ=UJO(LL)
      IF (INTLEV.EQ.1) GO TO 2505
      KSTOP=INTLEV-1
      DO 2500 K=1,KSTOP
          IF (ABS(SEPVAR(K)).NE.LL) GO TO 2500
          LJ=LOWER(K)
          UJ=UPPER(K)
2500 CONTINUE
2505 IF (UJ.LT.UJO(LL) .OR. LL.LT.NUMX1) GO TO 2510
      UP=0.0
      LO=0.0
      CALL XADDUB(BNDTYP,IOERR,PAIR(LL),LO,MAPI,MAPR,
1          MEMI,MEMR,UP)
2510 CALL XADDUB(BNDTYP,IOERR,LL,LJ,MAPI,MAPR,
1          MEMI,MEMR,UJ)
      SEPVAR(INTLEV)=0

```

```

      VXPLOR(INTLEV)=0
      INTLEV=INTLEV-1
      GO TO 2000
3000 MM=ABS(SEPVAR(INTLEV))
      INTNOD=INTNOD+1
      VXPLOR(INTLEV)=1
      KSTOP=INTLEV-1
      IF (MM.LT.NUMX1) GO TO 3200
      IF (SEPVAR(INTLEV).GT.0) GO TO 3300
      LO=0.0
      UP=UJO(PAIR(MM))
      CALL XADDUB(BNDTYP,IOERR,PAIR(MM),LO,MAPI,MAPR,
1          MEMI,MEMR,UP)
3200 IF (SEPVAR(INTLEV).GT.0) GO TO 3300
      SEPVAR(INTLEV)=MM
      LJ=0.0
      IF (INTLEV.EQ.1) GO TO 3255
      DO 3250 K=1,KSTOP
          IF (ABS(SEPVAR(K)).NE.MM) GO TO 3250
          LJ=LOWER(K)
3250 CONTINUE
3255 UJ=LOWER(INTLEV)-1.0
      IF (UJ.LT.0.0) UJ=0.0
      GO TO 3400
3300 LJ=UPPER(INTLEV)+1.
      UJ=UJO(MM)
      SEPVAR(INTLEV)=-MM
      IF (INTLEV.EQ.1) GO TO 3355
      DO 3350 K=1,KSTOP
          IF (ABS(SEPVAR(K)).NE.MM) GO TO 3350
          UJ=UPPER(K)
3350 CONTINUE
3355 IF (UJ.LT.UJO(MM) .OR. MM.LT.NUMX1) GO TO 3400
      LO=0.0
      UP=0.0
      CALL XADDUB(BNDTYP,IOERR,PAIR(MM),LO,MAPI,MAPR,
1          MEMI,MEMR,UP)
3400 CALL XADDUB(BNDTYP,IOERR,MM,LJ,MAPI,MAPR,
1          MEMI,MEMR,UJ)
      LOWER(INTLEV)=LJ
      UPPER(INTLEV)=UJ
      GO TO 3999
C      ***SELECT A BRANCHING VARIABLE WHEN INTEGER
C      REQUIREMENTS SATISFIED***
3500 K=1

```

```

      J=NUMINT
3510 IB=RANK(K)
      IF (IB.EQ.0) GO TO 2000
      CALL XGETUB(BNDTYP,IOERR,IB,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
      IF (LJ.NE.UJ) GO TO 3520
      WORK(J)=RANK(K)
      J=J-1
      K=K+1
      GO TO 3510
3520 IF (UJ.EQ.UBO(IB) .OR. CJ1(IB).LT.0.0) GO TO 3525
      IXBAS=UJ-1
      GO TO 3527
3525 IXBAS=LJ
3527 WORK(J)=RANK(K)
      L=1
3530 IF (L.EQ.J) GO TO 3540
      WORK(L)=RANK(K+1)
      L=L+1
      K=K+1
      GO TO 3530
3540 DO 3550 I=1,NUMINT
      RANK(I)=WORK(I)
3550 CONTINUE
      IF (UJ.EW.UBO(IB) .OR. CJ1(IB).GE.0.0) GO TO 102
      GO TO 105
3999 RETURN
4000 DONEIT=1
      RETURN
      END
      SUBROUTINE BILEVF (NUMX,NUMY,VARTYP,BIFEAS,UJO,
1      YCJ,BESTUB,BESTLB,CONX,CONY,
2      NSTRUC,CJ1,BSTCP,BSTNOD,XBEST,
3      BESTB,BESTAT,BESTLO,BESTUP,
4      ZBEST,INTNOD,B,BASCB,BASIS,
5      BASLB,BASUB,BLOW,ROWTYP,BNDTYP,
6      BOUND,COLA,COLI,COLMAX,IOERR,M,
7      MAPI,MAPR,MAXM,MAXN,MEMI,MEMR,
8      N,NTYPE2,SCODE,STATUS,UZERO,
9      XBZERO,Z,FACTOR,IOLOG,ITER1,
0      ITER2,LOOK,PICK,PRINT,TERMIN,
1      UNBDDQ,YQ)
C      *****PURPOSE
C      THIS SUBROUTINE IS USED TO FIX THE VARIABLES
C      CONTROLLED BY THE LEADER AND SOLVE THE RESULTING

```

```

C      MIXED INTEGER PROBLEM TO DETERMINE THE FOLLOWER'S
C      RESPONSE.
C
C      *****DECLARE XMP VARIABLES
      INTEGER COLMAX,MAXM,MAXN,PICK
      DOUBLE PRECISION XBZERO(MAXM),MEMR(LENR),B(MAXM),
1          BASCB(MAXM),BASLB(MAXM),
2          BASUB(600),BOUND,COLA(COLMAX),
3          UZERO(MAXM),YQ(MAXM),Z,BLOW(1)
      INTEGER FACTOR,IOLOG,ITER1,ITER2,LOOK,PRINT,
1          UNBDDQ,M,N,STATUS(MAXN),MAPI(LENI),
2          MAPR(LENMR),MEMI(LENI),BNDTYP,IOERR,
3          NSTRUC,NTYPE2,BASIS(MAXM),ROWTYP(MAXM),
4          TERMIN
C      *****DECLARE BLPP VARIABLES
      DOUBLE PRECISION XBZERO(MAXM),UJO(600),
1          BESTUB(600),BESTLB(600),
2          CJ1(600),XBEST(100),BESTLO(600),
3          BESTUP(600),ZBEST,ZMOST,YCJ(600)
      INTEGER BIFEAS,VARTYP(600),BESTB(100),
1          BESTAT(MAXN),NUMX,NUMY,CONX,CONY,
2          SCODE,COLI(COLMAX),INTNOD,BSTNOD,BSTCP,
      DOUBLE PRECISION CJ,BILO(600),BIUP(600),
1          BIBLO(600),BIBUP(600),
2          UPPR(600),LOWR(600),VARVAL,LJ,
3          UJ,DIFFR,YREM(100),BIZBST,
4          DUMXBZ(100),CHEC,DUMBUB(100),
      DUMBLB(100),BIBEST(100)
      INTEGER BILEV,BIPLOR(500),BISEP(500),BIBSTB(100),
1          BISTAT(600),YCNTR,DUMSTA(600),DUMBAS(100),
2          IBAS,IB,COLLEN
C      *****COMMON VARIABLES
      COMMON/XMPCOM/BIG,SMALL,ZL,ZLC,EPS1,
1          EPS2,EPS3,EPS4,EPS5,EPS6
      DOUBLE PRECISION BIG,SMALL,ZL,ZLC,EPS1,EPS2,
1          EPS3,EPS4,EPS5,EPS6
      COMMON/XMPLEN/LENI,LENMI,LENMR,LENR
      INTEGER LENI,LENMI,LENMR,LENR
C
C      ***INITIALIZE VARIABLES***
      BIZBST=-1.0D25
      Z=0.0
      BILEV=0
      ZMOST=0.0
      BIFEAS=0

```



```

      NUMXY=NUMX+NUMY
      NUMX1=NUMX+1
      NUMXY1=NUMXY+1
      NSTOPV=NUMXY+CONY
C     ***SAVE BOUNDS OF VARIABLES***
      DO 1100 I=1,NSTRUC
        CALL XGETUB(BNDTYP,IOERR,I,LJ,MAPI,MAPR,
1         MEMI,MEMR,UJ)
        UPPR(I)=UJ
        LOWR(I)=LJ
        IF (I.LE.NSTOPV) GO TO 1100
        BIBLO(I)=LJ
        BIBUP(I)=UJ
1100  CONTINUE
C     ***FREE ALL G'S AND U'S***
      DO 1200 I=NUMXY1,NSTRUC
        UJ=UJO(I)
        LJ=0.0
        CALL XADDUB(BNDTYP,IOERR,I,LJ,MAPI,MAPR,
1         MEMI,MEMR,UJ)
1200  CONTINUE
C     ***SAVE BASIS,STATUS,XBZERO***
      DO 1225 J=1,M
        DUMXBZ(J)=XBZERO(J)
        DUMBAS(J)=BASIS(J)
        DUMBUB(J)=BASUB(J)
        DUMBLB(J)=BASLB(J)
1225  CONTINUE
      DO 1275 J=1,N
        DUMSTA(J)=STATUS(J)
1275  CONTINUE
C     ***CHANGE OBJECTIVE FUNCTION COEFFICIENTS***
      DO 1400 I=1,NUMXY
        CJ=YCJ(I)
        CALL XCHGCJ(MAPI,MAPR,MEMI,MEMR,CJ,I,MAXN)
1400  CONTINUE
C     ***RESET THE BASCB ARRAY***
      DO 1450 I=1,M
        J=BASIS(I)
        CALL XGETAJ(CJ,COLA,COLI,COLLEN,COLMAX,IOERR,
1         J,MAPI,MAPR,MEMI,MEMR)
        BASCB(I)=CJ
1450  CONTINUE
C     ***RESET ALL BOUNDS ON Y***
      DO 1500 I=NUMX1,NUMXY

```

```

        LJ=0.0
        UJ=UJO(I)
        CALL XADDUB(BNDTYP,IOERR,I,LJ,MAPI,MAPR,
1          MEMI,MEMR,UJ)
1500 CONTINUE
C    ***FIX THE X'S AND REMEMBER THE Y'S***
5000 YCNTR=1
      DO 5100 I=1,NUMXY
        IF (DUMSTA(I)) 5200,5300,5400
5200      IF (DUMSTA(I).EQ.-2 .OR. DUMSTA(I).EQ.-3)
1        GO TO 5210
        IF (DUMSTA(I).EQ.-4) GO TO 5300
        VARVAL=BESTUB(I)
        GO TO 5490
5210      DO 5220 J=1,M
        IF (DUMBAS(J).NE.I) GO TO 5220
        VARVAL=DUMXBZ(J)
        GO TO 5490
5220      CONTINUE
5300      IF (BNDTYP.NE.4) GO TO 5310
        VARVAL=BESTLB(I)
        GO TO 5490
5310      VARVAL=0.0
        GO TO 5490
5400      IXX=DUMSTA(I)
        VARVAL=DUMXBZ(IXX)
5490      CONTINUE
        IF (I.GT.NUMX) GO TO 5495
        ZMOST=ZMOST+CJ1(I)*VARVAL
        CALL XADDUB(BNDTYP,IOERR,I,VARVAL,
1          MAPI,MAPR,MEMI,MEMR,UJO(I))
        GO TO 5100
5495      YREM(YCNTR)=VARVAL
        YCNTR=YCNTR+1
5100 CONTINUE
      GO TO 6900
C    ***SELECT A BRANCHING VARIABLE***
6000 CHEC=-100000.
      IB=0
      DO 6100 I=NUMX1,NUMXY
        IF (VARTYP(I).EQ.0 .OR. VARTYP(I).EQ.3) GO TO 6100
        IF (STATUS(I).LE.0) GO TO 6100
        JJ=STATUS(I)
        IBAS=DINT(XBZERO(JJ))
        DIFFR=XBZERO(JJ)-DFLOAT(IBAS)

```

```

      IF (DIFFR.LE.0.000001 .OR. DIFFR.GT.0.999999)
1      GO TO 6100
      IF (DIFFR.LT.CHEC) GO TO 6100
      IB=I
      IXBAS=IBAS
      CHEC=DIFFR
6100 CONTINUE
      IF (IB.EQ.0) GO TO 110
C      ***BRANCH ON SELECTED VARIABLE***
101 BILEV=BILEV+1
      CALL XGETUB(BNDTYP,IOERR,IB,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
      LJ=DFLOAT(IXBAS)+1.0
      CALL XADDUB(BNDTYP,IOERR,IB,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
      BIPLOR(BILEV)=0
      BISEP(BILEV)=-IB
      BILO(BILEV)=LJ
      BIUP(BILEV)=UJ
      GO TO 6900
110 CONTINUE
      IF (Z.LE.BIZBST) GO TO 6300
C      FEASIBLE SOLUTION FOUND
      DO 6200 J=1,M
          BIBEST(J)=XBZERO(J)
          BIBSTB(J)=BASIS(J)
6200 CONTINUE
      DO 6210 J=1,N
          BISTAT(J)=STATUS(J)
          IF (J.GT.NSTOPV) GO TO 6210
          CALL XGETUB(BNDTYP,IOERR,J,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
          BIBLO(J)=LJ
          BIBUP(J)=UJ
6210 CONTINUE
      BIZBST=Z
      IF (BILEV.EQ.0) GO TO 7000
C      ***BACKTRACK***
6300 IF (BILEV.EQ.0) GO TO 7000
      IF (BIPLOR(BILEV).EQ.0) GO TO 6500
      LL=ABS(BISEP(BILEV))
      LJ=0.0
      UJ=UJO(LL)
      IF (BILEV.EQ.1) GO TO 6405
      KSTOP=BILEV-1

```

```

DO 6400 K=1,KSTOP
  IF (ABS(BISEP(K)) .NE. LL) GO TO 6400
  LJ=BILO(K)
  UJ=BIUP(K)
6400 CONTINUE
6405 CALL XADDUB(BNDTYP,IOERR,LL,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
  BISEP(BILEV)=0
  BIPLOR(BILEV)=0
  BILEV=BILEV-1
  GO TO 6300
6500 MM=ABS(BISEP(BILEV))
  BISEP(BILEV)=MM
  LJ=0.0
  IF (BILEV.EQ.1) GO TO 6555
  KSTOP=BILEV-1
  DO 6550 K=1,KSTOP
    IF (ABS(BISEP(K)) .NE. ABS(BISEP(BILEV)))
1      GO TO 6550
    LJ=BILO(K)
6550 CONTINUE
6555 UJ=BILO(BILEV)-1.0
  CALL XADDUB(BNDTYP,IOERR,MM,LJ,MAPI,MAPR,
1      MEMI,MEMR,UJ)
  BILO(BILEV)=LJ
  BIUP(BILEV)=UJ
  BIPLOR(BILEV)=1
C    ***USE THE XMP SUBROUTINES TO SOLVE A LINEAR
C    PROGRAMMING PROBLEM USING THE PRIMAL
C    SIMPLEX METHOD***
6900 CALL XSTART(B,BASCB,BASIS,BASLB,BASUB,
1      BNDTYP,BOUND,
2      COLA,COLI,COLMAX,IOERR,
3      M,MAPI,MAPR,MAXM,MAXN,MEMI,MEMR,
4      N,NTYPE2,SCODE,STATUS,
5      UZERO,XBZERO,Z)
C
C    CALL XPRIML(B,BASCB,BASIS,BASLB,
1      BASUB,BNDTYP,BOUND,
1      COLA,COLI,COLMAX,
2      FACTOR,IOERR,IOLOG,ITER1,ITER2,
3      LOOK,M,MAPI,MAPR,MAXM,MAXN,MEMI,MEMR,
4      N,NTYPE2,PICK,PRINT,STATUS,TERMIN,
5      UNBDDQ,UZERO,XBZERO,YQ,Z)
  IF (TERMIN.NE.1) GO TO 6300

```

```

      GO TO 6000
C    ***CHECK IF A BILEVEL FEASIBLE SOLUTION FOUND***
7000 YCNTR=1
      DO 7100 I=NUMX1,NUMXY
          IF (BISTAT(I)) 7200,7300,7400
7200     IF (BISTAT(I).EQ.-2 .OR. BISTAT(I).EQ.-3)
1         GO TO 7230
          IF (BISTAT(I).EQ.-4) GO TO 7300
C    HERE FOR NON-BASIC VARIABLES AT THEIR UPPER BOUNDS
      VARVAL=BIBUP(I)
      GO TO 7500
7230     DO 7240 J=1,M
          IF (BIBSTB(J).NE.I) GO TO 7240
          VARVAL=BIBEST(J)
          GO TO 7500
7240     CONTINUE
7300     IF (BNDTYP.NE.4) GO TO 7310
          VARVAL=BIBLO(I)
          GO TO 7500
7310     VARVAL=0.0
          GO TO 7500
7400     IXX=BISTAT(I)
          VARVAL=BIBEST(IXX)
7500     DIFFR=VARVAL-YREM(YCNTR)
          IF (I.GT.NUMX) ZMOST=ZMOST+VARVAL*CJ1(I)
          IF (DABS(DIFFR).LT.0.000001) GO TO 7505
          BIFEAS=1
7505     YCNTR=YCNTR+1
7100     CONTINUE
          GO TO 9000
          BIFEAS=1
C    ***RESET BOUNDS***
9000     DO 9100 I=1,NSTRUC
          UJ=UPPR(I)
          LJ=LOWR(I)
          CALL XADDUB(BNDTYP,IOERR,I,LJ,MAPI,MAPR,
1              MEMI,MEMR,UJ)
9100     CONTINUE
C    ***RESET STATUS BASIS XBZERO***
      DO 9125 J=1,M
          BASIS(J)=DUMBAS(J)
          XBZERO(J)=DUMXBZ(J)
          BASUB(J)=DUMBUB(J)
          BASLB(J)=DUMBLB(J)
9125     CONTINUE

```

```

      DO 9175 J=1,N
        STATUS(J)=DUMSTA(J)
9175  CONTINUE
C    ***RESET OBJECTIVE FUNCTION COEFFICIENTS***
      DO 9200 I=1,NUMXY
        CJ=CJ1(I)
        CALL XCHGCJ(MAPI,MAPR,MEMI,MEMR,CJ,I,MAXN)
9200  CONTINUE
C    ***RESET THE BASCB ARRAY***
      DO 9250 I=1,M
        J=BASIS(I)
        CALL XGETAJ(CJ,COLA,COLI,COLLEN,COLMAX,IOERR,J,
1      MAPI,MAPR,MEMI,MEMR)
        BASCB(I)=CJ
9250  CONTINUE
C    ***UPDATE INCUMBENT SOLUTION***
      DIFFR=ZMOST-ZBEST
      IF (DIFFR.LE.0.000001) GO TO 9999
C*****
C*    ABOVE LINE USED IN ALGORITHMS 1, 3, 4, 5, 7, *
C*          AND 8 *
C*    FOR ALGORITHMS 2 AND 6 SUBSTITUTE *
C*    FOLLOWING LINE: *
C*    IF (DIFFR.LE.0.000001) GO TO 998 *
C*****
      DO 250 J=1,M
        XBEST(J)=BIBEST(J)
        BESTB(J)=BIBSTB(J)
250  CONTINUE
      DO 260 J=1,N
        BESTAT(J)=BISTAT(J)
        BESTLO(J)=BIBLO(J)
        BESTUP(J)=BIBUP(J)
260  CONTINUE
      BSTNOD=INTNOD
      ZBEST=ZMOST
      CALL CPTIME(BSTCP,ITIME)
C*****
C*    FOR ALGORITHMS 2 AND 6 ADD *
C*    FOLLOWING LINES HERE: *
C*    GO TO 9999 *
C*998 BIFEAS=2 *
C*****
9999  RETURN
      END

```

```

SUBROUTINE XCHGCJ(MAPI,MAPR,MEMI,MEMR,CJ,I,MAXN)
C *****PURPOSE
C CHANGES THE VALUE OF THE OBJECTIVE FUNCTION
C COEFFICIENTS.
C
DOUBLE PRECISION CJ,MEMR(LENR)
INTEGER MAPI(LENMI),MAPR(LENMR),MEMI(LENI),MR3,
1 MAXN
C *****COMMON VARIABLES
COMMON/XMPLEN/LENI,LENMI,LENMR,LENR
INTEGER LENI,LENMI,LENMR,LENR
MR3=MAPR(3)
CALL XDATA5(CJ,I,MEMR(MR3),MAXN)
RETURN
END
SUBROUTINE XDATA5(CJ,J,PROFIT,MAXN)
C *****PURPOSE
C CHANGES THE OBJECTIVE FUNCTION COEFFICIENTS IN THE
C XMP DATA STRUCTURE.
C
INTEGER MAXN
DOUBLE PRECISION CJ,PROFIT(MAXN)
PROFIT(J)=CJ
RETURN
END
SUBROUTINE XPRINT(BASIS,BNDTYP,BOUND,
1 IOERR,IOLOG,NUMXY,M,
2 MAPI,MAPR,MAXM,MAXN,MEMI,MEMR,
3 N,NTYPE2,STATUS,XBZERO,Z,
4 BESTLO,BESTUP)
C *****PURPOSE
C PRINTS OUT THE SOLUTION OF THE MIXED INTEGER BLPP.
C
C *****PARAMETERS
INTEGER MAXM,MAXN
DOUBLE PRECISION BOUND,MEMR(LENR),XBZERO(MAXM),Z
INTEGER BNDTYP,IOERR,IOLOG,M,N,NTYPE2
INTEGER MAPI(LENMI),MAPR(LENMR)
INTEGER BASIS(MAXM),MEMI(LENI),STATUS(MAXN)
C *****LOCAL VARIABLES
DOUBLE PRECISION LJ,UJ,VALUE,BESTLO(600),
1 BESTUP(600)
INTEGER IX,J
C *****COMMON VARIABLES
COMMON/XMPLEN/LENI,LENMI,LENMR,LENR

```

```

      INTEGER LENI,LENMI,LENMR,LENR
C
C      *****BODY OF PROGRAM (XPRINT)
      WRITE(IOLOG,901)
901  FORMAT(1H0,28H XPRINT...CURRENT SOLUTION)
      WRITE(IOLOG,902)M,N
902  FORMAT(1H0,I6,20H LINEAR CONSTRAINTS,,5X,I6,
1      10H VARIABLES)
      WRITE(IOLOG,903)
903  FORMAT(1H0,25H AN SB FOLLOWING STATUS ,
1      32HINDICATES A SUPER-BASIC VARIABLE)
C
      WRITE(IOLOG,904)
904  FORMAT(1H0,10H VARIABLE,5X,10H STATUS,5X,
1      15H VALUE)
C
C
C      PRINT OUT THE VALUES OF ALL OF THE BASIC VARIABLES
C      AND ALL OF THE NON-BASIC VARIABLES THAT ARE AT
C      NON-ZERO BOUNDS.
      DO 280 J=1,NUMXY
          IF(STATUS(J))200,250,260
200      IF(STATUS(J).EQ.-2 .OR. STATUS(J).EQ.-3)
1          GO TO 230
          IF(STATUS(J).EQ.-4)GO TO 250
C
C      HERE FOR NON-BASIC VARIABLES AT THEIR UPPER
C      BOUNDS.
          IF(BNDTYP.LT.3)GO TO 210
          VALUE=BESTUP(J)
          GO TO 270
210      IF(BNDTYP.LT.2)GO TO 220
          IF(J.GT.NTYPE2)GO TO 220
          VALUE=BOUND
          GO TO 270
220      WRITE(IOLOG,905) J
905      FORMAT(16H ERROR: VARIABLE,I6,
1          16H IS AT ITS UPPER,
2          20H BOUND OF +INFINITY.)
          GO TO 280
C
C      HERE FOR FREE AND ARTIFICIAL VARIABLES.
230      DO 240 I=1,M
          IF(BASIS(I) .NE. J)GO TO 240
          VALUE=XBZERO(I)

```



```

                GO TO 270
240      CONTINUE
                GO TO 280
C
C      HERE FOR NON-BASIC VARIABLES AT THEIR LOWER
C      BOUNDS.
250      IF(BNDTYP .NE. 4)GO TO 280
                VALUE=BESTLO(J)
                GO TO 270
C
C      HERE FOR THE BASIC VARIABLES (EXCEPT FREE AND
C      ARTIFICIALS)
260      IX=STATUS(J)
                VALUE=XBZERO(IX)
                IF(IX .LE. M)GO TO 270
C
C      HERE FOR SUPER-BASIC VARIABLES.
                WRITE(IOLOG,908)J,STATUS(J),VALUE
908      FORMAT(1X,I10,5X,I10,1X,2HSB,2X,D15.8)
                GO TO 280
C
270      WRITE(IOLOG,906) J,STATUS(J),VALUE
906      FORMAT(1X,I10,5X,I10,5X,D25.12)
C
280 CONTINUE
C
C      PRINT THE OBJECTIVE VALUE.
                WRITE(IOLOG,907) Z
907      FORMAT('  VALUE OF LINEAR OBJECTIVE ',
1          'FUNCTION=',D15.8)
C
                RETURN
                END

```

APPENDIX B

THE GENERAL MULTILEVEL PROGRAMMING PROBLEM

B.1 Notation and Definitions

The multilevel programming problem as viewed by player 1 can be thought of as a standard mathematical program whose feasible region has been augmented to include a series of implicitly defined constraints. The notation proposed by Bialas and Karwan (1982) will be used to clarify this interpretation and recursively build up a succession of feasible regions for each of the players beginning at the lowest level. This will ultimately lead to a reformulation of problem (2.1).

To begin, let (x^a, x^b) be an arbitrary partitioning of $x \in R^n$ and denote the maximization of a bounded function $f(x^a, x^b)$ over a compact region $S \subset R^n$ for a fixed value of x^a by

$$\max_{x^b} (f(x^a, x^b) : x^a) \quad (B.1)$$

In the case of bilevel programming, if we think of x^a as the vector under the control of player 1, then player 2 is faced with the parameterized problem given in (B.1).

Definition B.1: For the partition (x^a, x^b) , the set of rational reactions of f over S is given by: $W_f(S) \equiv \{x \in S : f(x) = \max_{x^b} (f(x^a, x^b) : x^a)\}$.

If player 1 selects \bar{x}^a and there exists an \bar{x}^b which uniquely maximizes $f(\bar{x}^a, x^b)$ for all $(\bar{x}^a, x^b) \in S$, then the induced mapping

$$\bar{x}^b = \Phi_f(\bar{x}^a)$$

provides the rational reaction for each \bar{x}^a . This suggests an alternative representation of $W_f(S)$ which can be expressed as

$$W_f(S) = \{x \in S : x^b = \Phi_f(x^a)\}.$$

In effect, $W_f(S)$ denotes the feasible set for player 1 when all the decision variables are under his control. The BLPP can thus be written as

$$\max [f^1(x) : x \in W_{f^2}(S \cap (X^1 \times X^2))]$$

which in part supports our initial interpretation. When

the solution to (B.1) is not unique, Φ_f becomes a point-to-point mapping and the above formulation must be generalized accordingly. In order to extend the development to p levels, suppose $S^p \subset \mathbb{R}^n$ is nonempty and compact, and let $f^1(x), \dots, f^p(x)$ be bounded functions defined over S^1 . Player p , at the lowest level in the hierarchy, must then solve

$$\begin{aligned} & \max_{x^p \in X^p} [f^p(x) : (x^1, \dots, x^{p-1})] \\ & \text{subject to} \quad x \in S^p \end{aligned} \quad (P^1)$$

which is a standard optimization problem parameterized by the decision vectors x^1, \dots, x^{p-1} of the respective higher level players. The set S^p is defined as the level- p feasible region while the rational reaction set

$$S^{p-2} \equiv W_{f^p}(S^p)$$

is termed the level- $(p-2)$ feasible region.

In order to facilitate the presentation, although most of our results will hold regardless, it will now be assumed that the induced mapping Φ_{f^p} exists and that similarly defined mappings Φ_{f^i} , $i=p-1, \dots, 2$ also exist. This rules out that case of multiple optimal solutions for lower level problems and hence, nonuniqueness of objective functions at higher levels. As demonstrated

by Bard and Falk (1982a), such a complication might preclude the existence of a solution for (2.1). With this assumption we can write the solution to p^p as

$$x^p = \Phi_{f^p}(x^{p-1}, \dots, x^1)$$

At level $p-1$ the problem is given by

$$\begin{aligned} \max_{x^{p-1} \in X^{p-1}} [f^{p-1}(x) : (x^{p-2}, \dots, x^1)] \\ \text{subject to } x \in S^{p-1} \end{aligned}$$

where x^p does not explicitly appear in the partition of x but is contained implicitly in the arguments of the objective function. That is we can rewrite $f^{p-1}(x)$ as follows

$$\hat{f}^{p-1}(x^{p-1}, \dots, x^1) \equiv f^{p-1}(\Phi_{f^p}, (x^{p-1}, \dots, x^1), x^{p-1}, \dots, x^1)$$

and make the appropriate substitutions in p^{p-1} .

Continuing in this manner the level-k feasible region will be defined inductively as

$$S^k \equiv W_{f^{k+1}}(S^{k+1})$$

with the associated problem at that level given by

$$\begin{aligned}
 & \max_{x^k \in X^k} [f^k(x) : (x^{k-1}, \dots, x^1)] \\
 & \text{subject to } x \in S^k
 \end{aligned}
 \tag{P^k}$$

This establishes the interdependence of each player's problem. For the following partition of $x \equiv (x^{k-}, x^k, x^{k+})$ where

$$\begin{aligned}
 x^{k-} &\equiv (x^p, \dots, x^{k+1}) \\
 x^{k+} &\equiv (x^{k-1}, \dots, x^1),
 \end{aligned}$$

player k 's problem can be interpreted as one parameterized in x^{k+} and implicit in x^{k-} . From this point of view, we offer a pair of definitions to further clarify the meaning of feasibility and optimality for the MLPP.

Definition B.2: A point $(\hat{x}^{i-}, \hat{x}^i)$ is said to be optimal with respect to a point \hat{x}^{i+} if $(\hat{x}^{i-}, \hat{x}^i) = \Phi_{f^i}(\hat{x}^{i+})$, where $\Phi_{f^i}(\hat{x}^{i+})$ is the rational reaction for problem P^i , $i=p, \dots, 2$. Such a point \hat{x} will be called feasible to the MLPP.

Definition B.3: A point x^* is said to be an optimal solution to the MLPP if

- a) x^* is feasible; and,
- b) for all feasible points $\hat{x} \in S(X)$, $f^1(x^*) \geq f^1(\hat{x})$, where $x = x^1 \times x^2 \times \dots \times x^p$.

Notice that part b) of Definition B.3 only requires that the highest level player's objective function be maximized; part a) assures that each of the lower level players achieves the best results possible in the face of higher level decisions. In fact it is quite likely that a feasible point \hat{x} will exist such that $f^i(\hat{x}) > f^i(x^*)$, $i \neq p$, for one or more players. Perhaps more vexing though is the likelihood that an infeasible point $\bar{x} \in S(X)$ will exist such that $f^i(\bar{x}) \geq f^i(x^*)$ for all i , implying that the optimal solution to the MLPP may not be Pareto-optimal.

B.2 Example B.1

$$\max_{x^1} (f^1 = 2x^2 - x^1) \quad \text{where } x^2 \text{ solves}$$

$$\max_{x^2} (f^2 = 10x^1 + 6x^2 - 1x^3) \quad \text{where } x^3 \text{ solves}$$

$$\max_{x^3} (f^3 = -x^1 + 12x^2 - x^3)$$

$$\begin{aligned} \text{subject to } & x^1 - 8x^2 + x^3 \geq -8 \\ & -3x^1 - 3x^2 + x^3 \geq -15 \\ & \quad -x^2 - 2x^3 \geq -12 \quad S^3 \\ & \quad -2x^2 + 3x^3 \geq 6 \\ & \quad 3x^1 - x^2 \geq 3 \\ & x^1 \geq 0, x^2 \geq 0, x^3 \geq 0 \end{aligned}$$

The three regions for this example are depicted in Figure B.1. The collective choices x^1 , x^2 , and x^3 must fall within the polyhedron S^3 which is the level-3 feasible region. From f^3 it can be seen that once x^1 and x^2 are chosen, player 3 will react by picking the smallest possible value of x^3 . This produces the

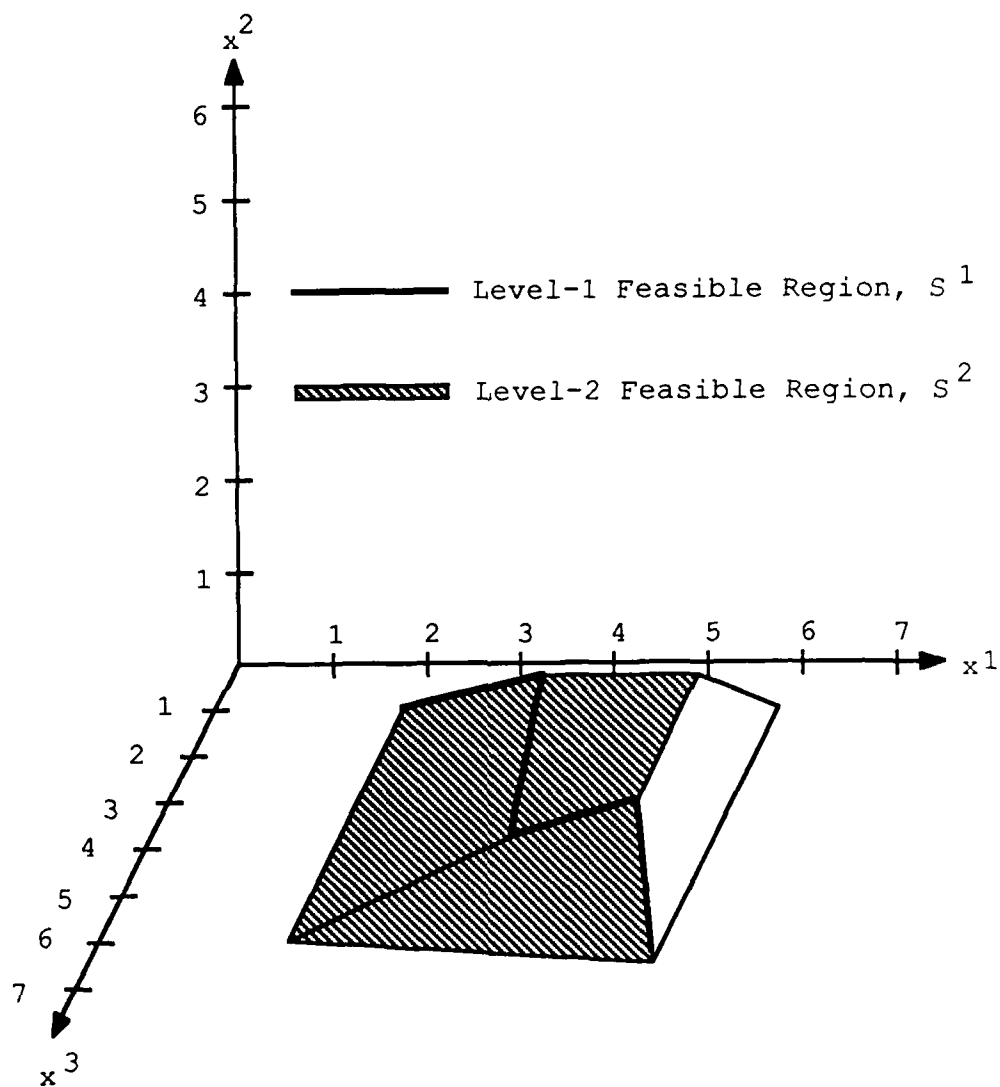


Figure B.1: Feasible Regions for Example B.1

rational reaction set S^2 (the level-2 feasible region) denoted by the hatched faces in Figure B.1. Thus, for x^1 fixed at say \bar{x}^1 , player 2's parametric problem is to maximize $f^2 = 10\bar{x}^1 + 6x^2 - x^3$ over

$$S^2 = \{x \in S^3 : f^3(x) = \max_{\hat{x}^3} (f^3 = -\bar{x}^1 + 12x^2 + x^3 : x^2)\}$$

Player 1's problem is then to maximize $f^1 = -x^1 + 2x^2$ over the second reaction set S^1 (the level-1 feasible region) which is marked by the emboldened edges.

Specifically, we have

$$S^1 = \{x \in S^3 : f^2(x) = \max_{\hat{x}^2} (f^2 = 10x^1 + 6\hat{x}^2 - x^3 : x^1), x^1 \in S^2\}$$

The optimal solution to the overall problem is found by solving the highest level problem, and is denoted by point H in Figure B.1. As expected, it occurs at a vertex of S^1 as well as S^3 . Point F, represents local optimum since movement in any feasible direction away from this point will produce a decrease in f^1 . Finally, an examination of S^3 reveals a Pareto-optimum at (the infeasible) point G; that is, a strict improvement in each objective function is realized when

we move from H to G. Table B.1 summerizes the results for these three situations.

TABLE B.1: OBJECTIVE FUNCTION VALUES OF VARIOUS OPTIMA

	Global Optimum, H	Local Optimum, F	Pareto Optimum, G
	value of x		
	(1.52, 1.57, 3.05)	(4.91, 2.27, 4.47)	(1.61, 1.84, 5.08)
f^1	1.52	-0.07	2.07
f^2	21.57	53.41	22.06
f^3	14.27	17.86	15.39

A basic assumption in multilevel programming rules out cooperation among the players. As demonstrated in Example B.1, if this restriction is relaxed, the payoff realized at each level may increase. The possibility of nonefficient solutions, though has wider implications for our work because it discourages the adaptation of algorithms designed to solve the vector maximization problem.

BIBLIOGRAPHY

Aiyoshi, E. and K. Shimizu, "Hierarchical Decentralized Systems and Its New Solution by a Barrier Method," IEEE Tran. Systems, Man., and Cybernetics SMC-11(6) 444-449 (June 1981).

Al-Khayyal, F.A. and J.E. Falk, "Jointly Constrained Biconvex Programs", Mathematics of Operations Research 8(2) 273-286 (1983).

Aoki, K. and T. Satoh, "Economic Dispatch with Network Security Constraints Using Parametric Quadratic Programming", IEEE Trans. Power Apparatus and Systems PAS-101(12) 4548-4556 (1982).

Bard, J.F., "Optimality Conditions for the Bilevel Programming Problem", Naval Research Logistics Quarterly 31 13-26 (1984).

Bard, J.F., "Regulating Nonnuclear Industrial Wastes by Hazard Classification", Journal of Environmental Systems 13(1) 21-41 (1983-1984).

Bard, J.F., "An Algorithm for Solving the General Bilevel Programming Problem", Mathematics of Operations Research 8(2) 260-272 (1983a).

Bard, J.F., "An Efficient Point Algorithm for a Linear Two-Stage Optimization Problem", Operations Research 31(4) 670-684 (1983b).

Bard, J.F., "Coordination of a Multidivisional Firm through Two Levels of Management", Omega 11(5) 457-468 (1983c).

Bard, J.F., "Geometric and Algorithmic Developments for a Hierarchical Planning Problem," European J. of Operational Research 19 372-383 (1985).

Bard, J.F., "Optimization in Multilevel Systems", Proceedings of the American Control Conference 1 403-408 Arlington, Virginia (1982).

Bard, J.F. and S. Chatterjee, "Objective Function Bounds for the Inexact Linear Programming Problem with Generalized Cost Coefficients", Computers and Operations Research 12(5) 483-491 (1985).

Bard, J.F. and J.E. Falk, "An Explicit Solution to the Multi-Level Programming Problem", Computers and Operations Research 9(1) 77-100 (1982a).

Bard, J.F. and J.E. Falk, "Necessary Conditions for the Linear Three Level Programming Problem", Proceedings of the 21th IEEE Conference on Decision and Control 2 642-646 Orlando (1982b).

Basar, T. and G.J. Olsder. Dynamic Noncooperative Games. Academic Press, New York (1982).

Basar, T. and H. Selbuz, "Closed Loop Stackelberg Strategies with Applications in Optimal Control of Multilevel Systems", IEEE Trans. Automatic Control AC-24(2) 166-178 (1979).

Baumol, W.J. and T. Fabian, "Decomposition, Pricing for Decentralization and External Economies", Management Science 11 1-32 (1964).

Baumol, W.J. and W.E. Oates. The Theory of Environmental Policy. Prentice-Hall, New Jersey (1975).

Bialas, W.F. and M.H. Karwan, "On Two-Level Optimization", IEEE Trans. Automatic Control AC-27(1) 211-214 (1982).

Bialas, W.F. and M.H. Karwan, "Two-Level Linear Programming", Management Science 30(8) 1004-1020 (1984).

Bialas, W.F., M.H. Karwan and J. Shaw, "A Parametric Complementary Pivot Approach to Multilevel Programming", Research Report No. 80-2, Dept. of Industrial Engineering, SUNY at Buffalo (1980).

Bitran, G.R., "Theory and Algorithms for Linear Multiple Objective Programs with Zero-One Variables," Mathematical Programming 17 362-390 (1979).

Bracken, J., J.E. Falk, and F.A. Miercort, "A Strategic Weapons Exchange Allocation Model," Operations Research 25 968-976 (1977).

Burton, R.M. and B. Obel, "The Multilevel Approach to Organizational Issues of the Firm: A Critical Review", Omega 5(4) 395-444 (1977).

Candler, W. and R. Townsley, "A Linear Two-Level Programming Problem", Computers and Operations Research 9(1) 59-76 (1982).

Cassidy, R., M.J. Kirby and W.M. Raikes, "Efficient Distribution of Resources through Three Levels of Government", Management Science 17(8) 462-473 (1971).

Chang, T.S. and P.B. Luh, "The Concept of Inducible Region in Stackelberg Games", Proceedings of the American Control Conference 1 139-140 Arlington, Virginia (1982a).

Chang, T.S. and P.B. Luh, "A Complete Solution for Two-Person, Single-Stage, Deterministic Stakelberg Games", Proceedings of the 21st IEEE Conference on Decision and Control 1 176-180 (1982b).

Charnes, A., R.W. Clower and K.O. Kortaner, "Effective Control through Coherent Decentralization with Preemptive Goals", Econometrica 35(2) 294-319 (1967).

Cruz, J.B., Jr., "Leader-Follower Strategies for Multilevel Systems", IEEE Trans. Automatic Control AC-23(2) 244-255 (1978).

Evans, B.C. and F.J. Gould, "Stability in Nonlinear Programming", Operations Research 18(1) 107-118 (1970).

Falk, J.E., "A Linear Max-Min Problem", Mathematical Programming 5 169-188 (1973).

Fiacco, A.V., "Sensitivity Analysis for Nonlinear Programming Using Penalty Methods", Mathematical Programming 10(3) 287-311 (1976).

Fiacco, A.V. and G. McCormick. Nonlinear Programming: Sequential Unconstrained Minimization Technique. Wiley, New York (1968).

Fortuny-Amat, J. and B. McCarl, "A Representation and Economic Interpretation of a Two-Level Programming Problem", Journal of the Operational Research Society 32 783-792 (1981).

Gallo, G. and A Ulkucu, "Bilinear Programming: An Exact Algorithm", Mathematical Programming 12 173-194 (1977).

Garfinkel, R.S. and G.L. Nemhauser, Integer Programming, John Wiley & Sons, NY (1972).

Gehner, K.J., "Necessary and Sufficient Conditions for the Fritz John Problem with Linear Constraints", SIAM J. Control 12(1) 140-149 (1974).

Geoffrion, A., "Primal Resource-Directive Approaches for Optimizing Non-Linear Decomposable Systems", Operation Research 18(3) (1970).

Geoffrion, A.M. and R.E. Marsten, "Integer Programming Algorithms: A Framework and State-Of-The-Art Survey," Management Science 18(9) 465-491 (May 1972).

Grotte, J.H., "Program MOOG - A Code for Solving Separable Non-Convex Optimization Problems", P-1318, The Institute for Defense Analyses, Arlington, Virginia (1976).

Haimes, Y.Y., W.A. Hall and H.T. Freedman. Multiobjective Optimization in Water Resources Systems. Elsevier, Holland (1975).

Hogan, W.W., "Point-To-Set Maps in Mathematical Programming," Siam Review 15(3) 591-603 (July 1973).

Jensen, P.A. Micrsolve/Operations Research: An Introduction to Operations Research with Microcomputers. Holden-Day, Oakland, California (1983).

Keeney, R.L. and H. Raiffa. Decisions with Multiple Objectives: Preferences and Value Trade-Offs. Wiley, New York (1976).

Konno, H., "A Cutting Plane Algorithm for Solving Bilinear Programs", Mathematical Programming 11 14-27 (1976).

Knuth, D.E. The Art of Computer Programming Vol 2. Addison-Wesley, Reading, Mass. (1969).

Lasdon, L. Optimization Theory for Large Systems. Macmillan, New York (1970).

Luce, R.D. and H. Raiffa. Games and Decisions. Wiley, New York (1957).

Luh, P.B., T.S. Chang and T.K. Ning, "Three-Level Hierarchical Decision Problems", Proceedings of the 5th MIT/ONR Workshop on Command, Control, and Communications Systems, Monterey, California (1982).

Mahoud, M.S. "Multilevel Systems Control and Applications: A Survey", IEEE Trans. Systems, Man, and Cybernetics SMC-7(3) 125-143 (1977).

Marsten, R.E., "The Design of the XMP Linear Programming Library," ACM Trans. Mathematical Software 7(4) 481-497 (Dec. 1981).

Mesarovic, M., D. Macko and Y. Takahara. Theory of Hierarchical Multi-Level Systems. Academic Press, London (1970).

Rockafellar, R.T. Convex Analysis. Princeton, New Jersey (1970).

Roodman, G.M., "Postoptimality Analysis in Zero-One Programming by Implicit Enumerations," Naval Logistics Quarterly 19(3) 435-447 (1972).

Schrage, L. and L. Wolsey, "Sensitivity Analysis for Branch and Bound Integer Programming," Operations Research 33(5) 1008-1023 (1985).

Simaan, M. and J.B. Cruz, Jr., "On Stackelberg Strategy in Nonzero-Sum Games," J. Optimization Theory and Applications 11 533-555 (1973).

Tarvainen, K. and Y.Y. Haimes, "Coordination of Hierarchical Multiobjective Systems: Theory and Methodology", IEEE Trans. Systems, Man, and Cybernetics SMC-7(3) 125-43 (1977).

Tolwinski, B., "Closed-Loop Stackelberg Solution to Multi-Stage Linear-Quadratic Game", Journal of Optimization Theory and Applications 34(4) 485-501 (1981).

Vaish, H. and C.M. Shetty, "The Bilinear Programming Problem", Naval Research Logistics Quarterly 23 303-309 (1976).

Vincent, T.L. and W.J. Grantham. Optimality in Parametric Systems. Wiley-Interscience, New York (1981).

Wen, U.P., "Mathematical Methods for Multilevel Linear Programming", Ph.D. Dissertation, Dept. of Industrial Engineering, SUNY at Buffalo (1981).

Winkofsky, E.P., N.R. Baker and D.J. Sweeney, "A Decision Process Model of R&D Resource Allocation in Hierarchical Organizations", Management Science 27(3) 268-283 (1981).

Yu, P.L., "Cone Convexity, Cone Extreme Points, and Nondominated Solutions in Decision Problems with Multiobjectives," J. Optimization Theory and Applications 14(3) 321-377 (1974).

Yu, P.L. and M. Zeleny, "The Set of all Nondominated Solutions in Linear Cases and Multicriteria Simplex Method", J Mathematical Analysis and Applications 49(2) 430-468 (1975).

Zangwill, W.L. and C.B. Garcia, "Equilibrium Programming: Path-Following Approach and Dynamics", Mathematical Programming 21 262-289 (1981).

Zeleny, M. Multiple Criteria Decision Making. McGraw-Hill, New York (1982).

Zionts, S., "Integer Linear Programming with Multiple Objectives," Annals of Discrete Mathematics 1 551-562 (1977).

VITA

PII Redacted

James Thomas Moore was born at [REDACTED]
[REDACTED] [REDACTED] to James H. and Lucille M.
Moore. After graduating from Mitchell High School in
Colorado Springs, Colorado in 1970, he attended the
University of Colorado. He graduated with a B.A. in
mathematics in 1974. After graduation, he joined the
United States Air Force and was commissioned in 1975.
In 1978, he was awarded an MBA by the University of
Wyoming. He graduated with an M.S in Operations
Research from the Air Force Institute of Technology in
1981. James entered the Graduate School of the
University of Texas in 1984.

PII Redacted

Permanent address: [REDACTED]

This dissertation was typed by James Thomas Moore.